Nonconvex Alternating Direction Optimization for Graphs : Inference and Learning

Đ.Khuê Lê-Huu

université PARIS-SACLAY

Inria inventeurs du monde numérique

CentraleSupélec

NNT : 2019SACLC005

Thèse de doctorat

Nonconvex Alternating Direction Optimization for Graphs: Inference and Learning

Thèse de doctorat de l'Université Paris-Saclay
préparée à CentraleSupélec

École doctorale n°580 Sciences et technologies de l'information et de la communication (STIC)

Spécialité : Traitement du signal et des images

Thèse présentée et soutenue à Paris, le 4 février 2019, par

Lê-Huu, D. Khuê

Composition du jury :

Isabelle Bloch	
Professeur, Télécom ParisTech, Université Paris-Saclay	Présidente
Florence Forbes	
Directrice de Recherche, Inria Grenoble Rhône-Alpes	Rapporteur
Nikos Komodakis	
Professeur, École des Ponts ParisTech	Rapporteur
Ramin Zabih	
Professeur, Cornell University & Google	Examinateur
Karteek Alahari	
Chargé de Recherche, HDR, Inria Grenoble Rhône-Alpes	Examinateur
Dimitris Samaras	
Professeur, Stony Brook University	Invité
Nikos Paragios	
Professeur, CentraleSupélec, Université Paris-Saclay	Directeur de thèse

To my parents, my sister, and my wife.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisor, Prof. Nikos Paragios, for many things. I still remember the day when I came from afar to meet him for the first time in his office and said: "I would like to be your PhD student!". I was at the time a young student impressed by his scientific achievements. Now looking back, my courage on that day was quite unusual, but it is still something I am proud of. Nikos then gave me the chance to prove myself via the MVA MSc program of CentraleSupélec, which I completed with success and then officially became his PhD student. I have always been grateful to him for giving me this opportunity. One thing I particularly like being Nikos' student was the freedom to choose research topics and the autonomy to follow those directions. This did not work sometimes for a student like me who is disorganized and who likes jumping randomly between ideas. Fortunately, he was there to guide and push me when it was needed. I would like to thank him again for his support, patience, and many pieces of advice over the last four years. And last, but not least, my wife and I would like to thank Nikos for allowing her to come with me in my office to study during my third year, and for his caring, advice and encouragement when she was applying to a graduate program in France.

I am extremely grateful to the members of my PhD committee, Profs. Ramin Zabih, Florence Forbes, Isabelle Bloch, Dimitris Samaras, Nikos Komodakis, and Karteek Alahari, for generously offering their time and support. Prof. Forbes and Prof. Komodakis spent their precious time reading the manuscript and giving encouraging reviews. Prof. Bloch kindly helped me a lot in the preparation of the defense at Télécom ParisTech. Prof. Zabih and Prof. Samaras were very supportive and always happy to help, despite their extreme time constraints. Prof. Alahari was always kind to provide support and encouragement. I must thank my advisor, Prof. Paragios, for making a great effort to invite such a distinguished committee, and for spending a lot of time on the organization of the defense. I would like to thank again all committee members for their insightful comments and questions during the defense. Their valuable feedback has not only helped me to improve the manuscript but also inspired me to further extend my work in different directions.

During my PhD, I had the opportunity to discuss with and to learn from many respected researchers. I would like to thank Profs. Jean-Christophe Pesquet, Iasonas Kokkinos, Matthew Blashko, Pawan Kumar, Dimitris Samaras, and Minh Hoai for the valuable discussions I had with them. I very much appreciate the support of Prof. Samaras and Prof. Hoai during my two-month visit at the Computer Vision Laboratory of Stony Brook University. I would also like to thank Prof. Paul-Henry Cournède for chairing my mid-term defense.

A lot of administrative paperwork happened during my PhD, both at CentraleSupélec and Inria. A huge thank to our team managers Natalia Leclercq, Jana Dutrey, and Alexandra Merlin for their valuable help on this time-consuming matter, so that I could focus on my research. In particular, Jana provided me with consistent assistance

over the last few months during the preparation of my defense. I also gratefully acknowledge the help of Prof. Gilles Duc and Anne Batalie on paperwork issues related to the doctoral school.

Many thanks to all my friends and colleagues at the Center for Visual Computing for the wonderful time we spent together (in alphabetical order): Alp, Enzo, Eugene, Evgenios, Hari, Jiaqian, Maria, Marie-Caroline, Maxim, Mihir, Puneet, Rafael, Siddhartha, Stavros, Stefan, and Wacha. Thank you Maxim for kindly helping me to correct my French writing on multiple occasions. Special thanks to Maria, Siddhartha, and especially Mihir for their technical support related to our computing servers. Special thanks also to Marie-Caroline for kindly helping me multiple times on various things, such as turning on my lab desktop after a power outage, or bringing back documents that I needed urgently, while I was not able to come to the lab. Thanks also to Vu Nguyen, Tan Nhat Le, and other members of the Stony Brook Computer Vision Lab for their help during my visit there.

Several months before my graduation, I became a full-time researcher at Qopius Technology, a computer vision startup based in Paris. I would like to thank my CEOs Antonin Bertin and Roy Moussa for being supportive and flexible with my working hours so that I could smoothly complete the administrative procedure required for the defense. I must also thank my friend and colleague Victor Journé for his help on translating the abstract of this thesis into French. All possible remaining errors, however, are due to my deficiency. Thanks also to all my colleagues at Qopius for their encouragement during the preparation of the defense.

For me this thesis is the result of a long journey, started upon my arrival in France more than a decade ago. Therefore, I would like to take this opportunity to express my appreciation to the people and organizations who have helped and supported me since my early days in France. Foremost, I would like to extend my deepest gratitude to Prof. Jean Trần Thanh Vân and his wife, Prof. Lê Kim Ngọc, for bringing me to France and for taking care of me from the first day until now. They are two such great persons that I admire the most, and what they have done for me and for my country cannot be described in just a few words. I am also grateful to Prof. Đỗ Trinh Huệ for putting his trust in me and for his encouragement over the years. I am deeply indebted to Prof. Odon Vallet and his Vallet Foundation for generously offering me multiple scholarships for my undergraduate studies. During this time I also received financial support from the Region of Centre Val de Loire, the City of Blois, and my school ENIVL (known as INSA Centre Val de Loire since 2012). My first days in France would have been much more difficult without the help and support that I received from faculty members and classmates at ENIVL. I would like to thank them all again. In particular, I would like to express my special thanks to the following persons for their consistent help and caring over the years: Profs. Jérôme Fortineau, Romuald Boné, Pascal Trân, and faculty members Béatrice Trombetta, Nadège Courbois, Karine Goux-Brunet. I have been lucky to meet and become friends with very kind French families, whose support cannot be overestimated: Didier and Yvette Chaux; Pierre and Nicole Louis; Frédéric and Hélène Herlin; and especially, Hugues, Isabelle and Solène Vassal. Finally, I cannot end this paragraph without mentioning the following Vietnamese persons in Blois, who helped me a lot and who always made me feel at home: bà Yến, bác Hùng o Hương, bác Bảo bác Hẹ, and their families.

I would like to acknowledge the encouragement from my Vietnamese friends in France: cô Thao, chị Bê, Tuân, Lu, anh Tiến Anh bé Ly, anh Song chị Linh, anh em Kchan. Especially helpful to me during this time were my "nhà Blois" brothers and sisters who shared with me a lot of good and bad moments: Long, Rô Thuý, Tâm Chi, Cơ, Thuỷ, Linh, Minh, Thọ Linh, Tú Chi, Hải, Lâm Phương, Hảo, Phúc, Nhật. My special thanks go to Cơ and Hảo for their help during the preparation of the defense.

Over the years, I have luckily received tremendous love and support from Vietnam. I would like to thank all the people, relatives, teachers, and friends who cared about me. Special thanks to the following persons whom I consider to be family: bác Hoà cô Phương, bác Tùng o Nga, thầy Từ, chị Trang anh Tiến, anh Phương chị Hiền, and my best friends: Thoại, Cu Em, Trà Ngân, Trùm, Hào.

My family in Vietnam has always supported and nurtured me. I would like to thank my grandmothers, uncles, aunts, and cousins: Mệ Nội và Mệ Ngoại, gia đình bác Thuỳ bác Hiền, gia đình chú Tí, gia đình cậu Quốc dì Hoa, gia đình cậu Bi mợ Truyền, gia đình anh Phú, Luna của cậu; và Bà Ngoại, ba Quang mẹ Thảo và Bin, gia đình cậu Nguyên, gia đình mợ Mượn, cùng toàn thể gia đình bên ngoại.

Finally, I would like to dedicate this thesis to the most important persons in my life. To my parents, who have sacrificed all their life for me and my sister. To my sister, who loves me unconditionally. And to my wife Phương Hằng, who pours her life into mine. No words can describe how much I love them.

Paris, January 2019
Lê-Huu, D. Khuê

Abstract

This thesis presents our contributions to inference and learning of graph-based models in computer vision.

First, we propose a novel class of decomposition algorithms for solving graph and hypergraph matching based on the nonconvex alternating direction method of multipliers (ADMM). These algorithms are computationally efficient and highly parallelizable. Furthermore, they are also very general and can be applied to arbitrary energy functions as well as arbitrary assignment constraints. Experiments show that they outperform existing state-of-the-art methods on popular benchmarks.

Second, we propose a nonconvex continuous relaxation of maximum a posteriori (MAP) inference in discrete Markov random fields (MRFs). We show that this relaxation is tight for arbitrary MRFs. This allows us to apply continuous optimization techniques to solve the original discrete problem without loss in accuracy after rounding. We study two popular gradient-based methods, and further propose a more effective solution using nonconvex ADMM. Experiments on different real-world problems demonstrate that the proposed ADMM compares favorably with state-of-the-art algorithms in different settings.

Finally, we propose a method for learning the parameters of these graph-based models from training data, based on nonconvex ADMM. This method consists of viewing ADMM iterations as a sequence of differentiable operations, which allows efficient computation of the gradient of the training loss with respect to the model parameters, enabling efficient training using stochastic gradient descent. At the end we obtain a unified framework for inference and learning with nonconvex ADMM. Thanks to its flexibility, this framework also allows training jointly end-to-end a graph-based model with another model such as a neural network, thus combining the strengths of both. We present experiments on a popular semantic segmentation dataset, demonstrating the effectiveness of our method.

Résumé

Cette thèse présente nos contributions à l'inférence et l'apprentissage des modèles graphiques en vision artificielle.

Tout d'abord, nous proposons une nouvelle classe d'algorithmes de décomposition pour résoudre le problème d'appariement de graphes et d'hypergraphes, s'appuyant sur l'algorithme des directions alternées (ADMM) non convexe. Ces algorithmes sont efficaces en terme de calcul et sont hautement parallélisables. En outre, ils sont également très généraux et peuvent être appliqués à des fonctionnelles d'énergie arbitraires ainsi qu'à des contraintes de correspondance arbitraires. Les expériences montrent qu'ils surpassent les méthodes de pointe existantes sur des benchmarks populaires.

Ensuite, nous proposons une relaxation continue non convexe pour le problème d'estimation du maximum a posteriori (MAP) dans les champs aléatoires de Markov (MRFs). Nous démontrons que cette relaxation est serrée, c'est-à-dire qu'elle est équivalente au problème original. Cela nous permet d'appliquer des méthodes d'optimisation continue pour résoudre le problème initial discret sans perte de précision après arrondissement. Nous étudions deux méthodes de gradient populaires, et proposons en outre une solution plus efficace utilisant l'ADMM non convexe. Les expériences sur plusieurs problèmes réels démontrent que notre algorithme prend l'avantage sur ceux de pointe, dans différentes configurations.

Finalement, nous proposons une méthode d'apprentissage des paramètres de ces modèles graphiques avec des données d'entraînement, basée sur l'ADMM non convexe. Cette méthode consiste à visualiser les itérations de l'ADMM comme une séquence d'opérations différenciables, ce qui permet de calculer efficacement le gradient de la perte d'apprentissage par rapport aux paramètres du modèle. L'apprentissage peut alors utiliser une descente de gradient stochastique. Nous obtenons donc un framework unifié pour l'inférence et l'apprentissage avec l'ADMM non-convexe. Grâce à sa flexibilité, ce framework permet également d'entraîner conjointement de-bout-en-bout un modèle graphique avec un autre modèle, tel qu'un réseau de neurones, combinant ainsi les avantages des deux. Nous présentons des expériences sur un jeu de données de segmentation sémantique populaire, démontrant l'efficacité de notre méthode.

Contents

List of Figures

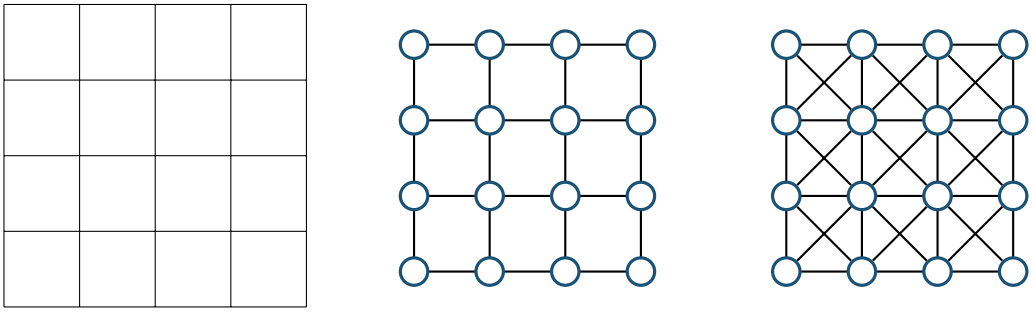
List of Tables

List of Algorithms

1

Introduction

Over the last decades, graph-based representations have become a ubiquitous tool for solving a wide range of problems in computer vision and pattern recognition. One of the main reasons for this success is that such representations are very natural and powerful for modeling structural and contextual relationships, which are essential in many visual perception tasks, from low-level (*e.g.* segmentation, denoising, filtering, *etc.*) to high-level vision (*e.g.* object recognition, scene understanding, pattern matching, *etc.*). For example, to model an image at the low-level, one can use a graph whose nodes are the image pixels and whose edges represent the neighbor relationships between them (as illustrated in Figure 1.1); or at the high-level, an object in that image can be represented by a graph whose nodes are object parts and whose edges represent the connections between these parts.



(a) An image with 4×4 pixels. (b) 4-connected graph. (c) 8-connected graph.

FIGURE 1.1 An example of graph-based representations for images. The left figure is an illustration of an image with 4×4 pixels. For different vision tasks such as segmentation or denoising, one can model this image using a graph where the nodes represent the pixels and the edges represent the neighboring relationships between them. The middle and the right figures show two different usual neighboring systems.

Perhaps the most popular and dominant graph-based representation in computer vision to date is *Markov random fields* (MRFs), a special class of *probabilistic graphical models*. In a few words, an MRF is a model that uses an undirected graph to compactly encode a family of joint probability distributions, where the graph nodes represent the corresponding random variables, and the graph structure represents the probabilistic interactions between these variables. With suitable modeling, many computer vision tasks can be reduced to solving the so-called *maximum a posteriori (MAP) inference* problem over the underlying joint distributions, which consists of finding the most likely joint assignment to the random variables. MAP inference can be reformulated as minimizing a function, called the *energy* of the given MRF. Therefore, this problem

is also referred to as *MRF energy minimization* in the literature. In this thesis, we use "MAP inference" and "energy minimization" interchangeably.

The very first application of MRFs in computer vision and image processing was proposed in the seminal paper by [Geman and Geman, 1987] for image denoising and restoration. Nevertheless, MRFs only really took off more than a decade later, when a very significant progress was achieved in [Boykov et al., 1999] for efficiently (and approximately) minimizing a special class of energy functions. Indeed, the major difficulty with MRF energy minimization for vision applications lies in the enormous computational costs. Theoretically, this problem is known to be NP-hard for general energy functions [Shimony, 1994]. In practice, worse still, problem sizes in computer vision are typically very large, which makes energy minimization further intractable. The method proposed in [Boykov et al., 1999], based on *graph cuts*, can efficiently produce a solution within a known approximation bound of the global optimum. Although it can only be applied to a certain class of functions[1], this class is general enough to include a wide range of models arising in many vision applications. However, it remains a special class after all, and thus, has a major limitation in terms of modeling capability. With a more general energy function, the corresponding MRF has more expressive power, but at the same time, it is more difficult to be optimized. As a consequence, finding novel efficient optimization methods for more general MRFs have become a very active research topic over the last two decades. Many MRF optimization methods have been proposed in the literature and can be roughly grouped into two classes:[2] (a) methods that stay in the discrete domain, such as move making and belief propagation [Boykov et al., 2001, Yedidia et al., 2005, Komodakis et al., 2008, Fix et al., 2011], and (b) methods that move into the continuous domain by solving convex relaxations such as quadratic programming relaxations [Ravikumar and Lafferty, 2006], semi-definite programming relaxations [Olsson et al., 2007], or most prominently linear programming (LP) relaxations [Wainwright et al., 2005, Kolmogorov, 2006, Globerson and Jaakkola, 2008, Komodakis et al., 2011], *etc.* On one hand, discrete methods tackle directly the original problem, which is very challenging because of its combinatorial nature, thus many of them are only applicable to certain classes of the energy. Convex relaxation methods, on the other hand, can be applied to minimizing general energy functions; in addition, they allow us to benefit from the tremendous convex optimization literature. Thanks to the convexity, these relaxations can be solved exactly in polynomial time, but they often only produce real-valued solutions that, after a rounding step, can reduce significantly the accuracy if they are not tight.

Another major graph-based representation is *graph matching*, used primarily for solving the correspondence problem. Finding correspondences between two sets of objects (*e.g.* feature points, object parts, *etc.*) is a fundamental problem that has a wide range of applications in computer vision and pattern recognition. Examples include depth estimation, 3D reconstruction, object detection, shape matching, image registration, *etc.* This problem plays a pivotal role in computer vision, and as a consequence, graph matching is of fundamental importance. The general idea of solving the corre-

[1] The method proposed in [Boykov et al., 1999] is called α-expansion and can be applied to minimizing energy functions whose second-degree terms, called *pairwise potentials*, are a metric, *i.e.* a function $f(\alpha, \beta)$ satisfying three conditions: (a) $f(\alpha, \beta) = f(\beta, \alpha) \geq 0$, (b) $f(\alpha, \beta) = 0$ if and only if $\alpha = \beta$, and (c) $f(\alpha, \beta) \leq f(\alpha, \gamma) + f(\gamma, \beta)$ for any α, β, γ in the label (or state) space. The relevant definitions will be presented in Chapter 2.

[2] We discuss these methods later in Chapter 2, with a more detailed classification.

spondence problem via graph matching is to associate each set of objects an attributed graph, where the node attributes describe local characteristics while the edge ones describe structural or contextual relationships. The matching task seeks to minimize an objective function that represents the differences between the corresponding nodes as well as the corresponding edges. For a better modeling capability, the edges can be generalized to be subsets of more than two nodes, called *hyperedges*, and in this case the problem is called *hypergraph matching* or *higher-order matching* (as such, *pairwise matching* refers to matching regular graphs). In its general form, the objective function in graph matching is very similar to the one in MRF optimization, and is also called the *energy*. In graph matching, however, the constraints on the assignments can be more complex than in MRFs: depending on the application, a node from one set can have one or many correspondences in the other, which should be taken into account properly. When the energy is of second degree and the assignment obeys the one-to-one constraint, graph matching becomes the *quadratic assignment problem* (QAP), which is a very difficult combinatorial problem. It is known that the QAP is not only NP-hard but also NP-hard to approximate [Burkard et al., 1998]. Moreover, it is also practically intractable: problems with sizes larger than 20 are already considered to be "large-scale" in the combinatorial optimization literature [Burkard et al., 1998]. With the typically-large problem sizes in computer vision, it is even more challenging as most of existing combinatorial methods become impractical. This hard problem has been an active research topic in the computer vision field over the last two decades. Some of the most prominent works include [Gold and Rangarajan, 1996, Leordeanu and Hebert, 2005, Cour et al., 2007, Leordeanu et al., 2009, Cho et al., 2010], *etc.* for solving pairwise graph matching, and [Zass and Shashua, 2008, Duchenne et al., 2011, Lee et al., 2011, Nguyen et al., 2015], *etc.* for solving hypergraph matching. These methods all have their limitations: some only work for certain degrees of the energy (*e.g.* lower than fourth), some only work for certain types of assignments (*e.g.* one-to-one), some only work for non-negative energies, *etc.*

In recent years the field has seen an enormous increase in availability of data and computing power, and consequently, in problem sizes. This is the case not only in computer vision but also in machine learning, pattern recognition and other computational fields. As a result, there has been a need for algorithms that can be run in a parallel and distributed manner, so that large-scale problems can be efficiently solved by effectively exploiting the available computing resources. One of the most prominent approaches is to apply *decomposition methods* in optimization. Decomposition is a general approach to solving a problem by breaking it up into smaller ones that can be efficiently addressed separately (and possibly in parallel), and then reassembling the results towards a globally consistent solution of the original non-decomposed problem [Bertsekas, 1999]. In computer vision, decomposition methods have been applied to solving graph related problems such as MAP inference and graph matching, using *dual decomposition* [Komodakis et al., 2011, Torresani et al., 2013] or *alternating direction method of multipliers* (ADMM) [Martins et al., 2015]. The main idea is to decompose the original complex graph into simpler subgraphs and then reassembling the solutions on these subgraphs using different mechanisms. Both dual decomposition and ADMM originate from duality theory in *convex* optimization and thus they share some similarities, but ADMM has better convergence properties, both theoretically and practically. Originally introduced more than 40 years ago independently by [Glowinski and Marroco, 1975] and [Gabay and Mercier, 1975], ADMM has only

recently started to gain popularity in the machine learning and computer vision fields after the publication of an influential paper by [Boyd et al., 2011], showing that the method is indeed very flexible and powerful for solving large-scale problems.

In view of the above observations, in this thesis, we make three major contributions.

First, we propose a novel class of decomposition algorithms for solving graph and hypergraph matching based on nonconvex ADMM. Not only these methods are computationally efficient and achieve state-of-the-art accuracy on popular benchmarks, they are also very general: they work with arbitrary energy functions (of any degrees and any types), and with any assignment constraints (one-to-one, many-to-many, and the like), thus overcoming all the aforementioned limitations of existing methods.

Second, we propose a nonconvex continuous relaxation of MAP inference in discrete MRFs. We show that this relaxation is tight for arbitrary MRFs. Therefore, solving the new continuous relaxation is equivalent to solving the original discrete problem, which opens the door to applications of continuous optimization methods. We study two gradient-based methods, and further propose a more effective solution using a multilinear decomposition framework based on ADMM. Experiments on different real-world problems demonstrate that the proposed ADMM compares favorably with state-of-the-art algorithms in different settings.

Finally, we propose a method for learning the parameters of these graph-based models from training data, based on nonconvex ADMM. This method consists of viewing ADMM iterations as a sequence of differentiable operations, which allows efficient computation of the gradient of the training loss with respect to the model parameters, enabling efficient training using stochastic gradient descent. At the end we obtain a unified framework for inference and learning with nonconvex ADMM. Thanks to its flexibility, this framework also allows training jointly end-to-end a graph-based model with another model such as a neural network, thus combining the strengths of both. We present experiments on a popular semantic segmentation dataset, demonstrating the effectiveness of our method.

We start the thesis by three review chapters, which provide the mathematical background on the three key subjects of the thesis: MAP inference in MRFs in Chapter 2, graph and hypergraph matching in Chapter 3, and ADMM in Chapter 4. Our major contributions are then presented in the subsequent chapters. We present our novel graph matching framework in Chapter 5. Our proposed nonconvex relaxation for MAP inference, together with its resolution, is presented in Chapter 6. A unified framework for inference and learning with nonconvex ADMM is presented in Chapter 7. Finally, we discuss and conclude the thesis in the last chapter.

2

Inference in Markov Random Fields

In this chapter, we review Markov random fields and the problem of *maximum a posteriori* (MAP) inference in these models. We start by presenting the mathematical foundation of MRFs, and then review popular methods for solving MAP inference.

2.1 FOUNDATION OF MARKOV RANDOM FIELDS

Markov random fields (MRFs) are a special class of a more general paradigm called *probabilistic graphical models*, or *graphical models* for short. Although general graphical models are not the focus of the thesis, we find them to be essential to understand the mathematical foundation of MRFs. Therefore, in this section we give a brief introduction to them. For a more in-depth and complete presentation, we refer to the excellent book of [Koller and Friedman, 2009].

A probabilistic graphical model is a graph-based representation that can compactly encode a family of complex probability distributions over a high-dimensional space. In this representation, the graph nodes represent the corresponding random variables, and the graph structure represent probabilistic interactions (*i.e.* dependencies or independencies) between them. Depending on the type of the underlying graph, graphical models can be divided into two main classes: models using directed acyclic[1] graphs are called **Bayesian networks** (or *directed graphical models*), and those using undirected graphs are called **Markov networks** or **Markov random fields** (or *undirected graphical models*). Directed graphs are useful for expressing causal relationships between random variables, whereas undirected graphs are more suitable for expressing soft constraints between them. Examples of these models are given in Figure 2.1.

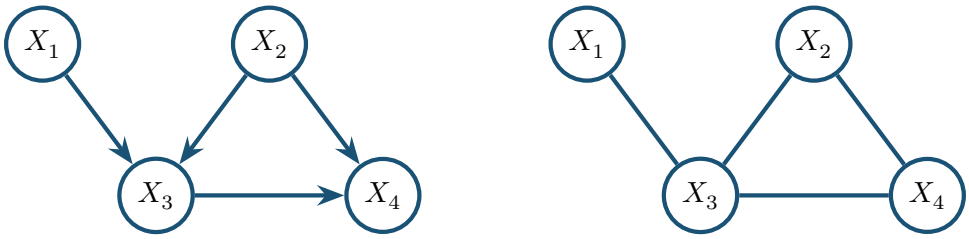
As we will see next, the two classes of graphical models have different ways to express independencies as well as to represent the underlying joint distribution.

2.1.1 *Local independence and distribution factorization*

To keep the presentation simple, let us restrict ourselves to the case of *discrete* random variables, *i.e.* those having a countable[2] number of possible outcomes. For a (discrete) random variable X, let $p(x)$ denote the probability mass function of the distribution over X, *i.e.* $p(x) = \Pr(X = x)\ \forall x$. Similar notation is used for joint or conditional distributions. Let us remind that for random variables X, Y and Z, we say that X

[1] There is no probabilistic graphical model defined on directed *cyclic* graphs.

[2] The term *countable* means either *finite* or *countably infinite*.



(a) A Bayesian network. (b) A Markov random field.

FIGURE 2.1 Examples of probabilistic graphical models over four random variables X_1, X_2, X_3 and X_4. Directed graphs are useful for expressing causal relationships between these random variables, whereas undirected graphs are more suitable for expressing soft constraints between them.

and Y are *(conditionally) independent* given Z, denoted $X \perp\!\!\!\perp Y \mid Z$, if

$$\Pr(X = x, Y = y \mid Z = z) = \Pr(X = x \mid Z = z) \Pr(Y = y \mid Z = z) \quad \forall x, y, z. \tag{2.1}$$

Note that the above can be written simply as $p(x, y \mid z) = p(x \mid z)p(y \mid z)$, using our aforementioned notation.

Now consider the joint distribution $p(\mathbf{X}) = p(X_1, X_2, \dots, X_n)$ and let $\mathcal{G}$ be a graph with n nodes representing $X_1, X_2, \dots, X_n$.

Bayesian networks

Suppose that $\mathcal{G}$ is directed and acyclic, then a Bayesian network associated with $\mathcal{G}$ encodes the following *local independencies*:

$$X_i \perp\!\!\!\perp \mathbf{X}_{\mathcal{ND}_i} \mid \mathbf{X}_{\mathcal{P}_i} \quad \forall i, \tag{2.2}$$

where $\mathcal{ND}_i$ and $\mathcal{P}_i$ denotes respectively the set of non-descendants and the set of parents of node i in $\mathcal{G}$. In other words, the local independencies state that each node is conditionally independent of its non-descendants given its parents. As such, the graph structure can be seen as an independency map of the nodes.

The local independencies (2.2) can lead to a nice factorization of the joint distribution over the variables. Indeed, if $p(\mathbf{X})$ satisfies (2.2) then it can be written as

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i | \mathbf{x}_{\mathcal{P}_i}) \quad \forall \mathbf{x}. \tag{2.3}$$

We say that $p(\mathbf{X})$ *factorizes according to* $\mathcal{G}$.

To see why (2.2) implies (2.3), let us consider for example a very simple Bayesian network of three nodes as shown in Figure 2.2. Clearly, according to (2.2), this network encodes the following local independency: $X_3 \perp\!\!\!\perp X_1 \mid X_2$. By the chain rule we have:

$$p(x_1, x_2, x_3) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2). \tag{2.4}$$

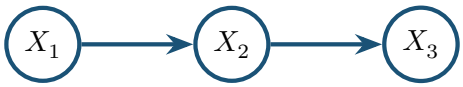


FIGURE 2.2 A simple Bayesian network. This network encodes the local independency $X_3 \perp\!\!\!\perp X_1 \mid X_2$.

Since $X_3 \perp\!\!\!\perp X_1 \mid X_2$ we have $p(x_3 \mid x_1, x_2) = p(x_3 \mid x_2)$ and thus (2.4) becomes

$$p(x_1, x_2, x_3) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2), \tag{2.5}$$

which is exactly (2.3) for the considered graph. The idea for a general graph is similar.

In fact, local independence and distribution factorization have an even tighter connection: *a distribution factorizes over a directed acyclic graph $\mathcal{G}$ if and only if the local independencies encoded by $\mathcal{G}$ hold in that distribution.* A proof can be found in [Koller and Friedman, 2009], together with more details on other types of independence that a Bayesian network can encode.

Markov random fields

Suppose that $\mathcal{G}$ is undirected, then a Markov random field associated with $\mathcal{G}$ encodes the following local independencies:

$$X_i \perp\!\!\!\perp \mathbf{X}_{\mathcal{V}\setminus\{i\cup\mathcal{N}_i\}} \mid \mathbf{X}_{\mathcal{N}_i} \quad \forall i, \tag{2.6}$$

where $\mathcal{N}_i$ denotes the set of neighboring nodes of i, and $\mathcal{V}\setminus\{i\cup\mathcal{N}_i\}$ denotes the set of nodes other than i and its neighbors. The above local independencies state that each node is independent of the rest of the nodes in the graph, given its neighbors. As a result, the set of neighbors of a node is also called the *Markov blanket* of that node.

Denote by $\mathcal{C}$ the set of *cliques* of $\mathcal{G}$ (a clique is a subset of fully connected nodes), then we say that $p(\mathbf{X})$ factorizes according to $\mathcal{G}$ if and only if $p(\mathbf{x})$ can be written as

$$p(\mathbf{x}) = \frac{1}{Z}\prod_{C\in\mathcal{C}} \psi_C(\mathbf{x}_C) \quad \forall \mathbf{x}, \tag{2.7}$$

where ψ_C is a non-negative function of the variables $\mathbf{x}_C = (x_i)_{i\in C}$ in the clique C, and $Z = \sum_{\mathbf{x}}\prod_{C\in\mathcal{C}}\psi_C(\mathbf{x}_C)$ is a normalization term so that the left-hand side of (2.7) is a valid probability distribution. The term Z is called *partition function* and $(\psi_C)_{C\in\mathcal{C}}$ are called *potential functions*. For a clique C, if its size $|C|$ is, respectively, 1 or 2 or > 2, then ψ_C is called, respectively, *unary* (or *singleton*) or *pairwise* or *higher-order* potential.

Similar to Bayesian networks, conditional independence and distribution factorization in Markov random fields are also highly related: *a positive distribution factorizes over an undirected graph $\mathcal{G}$ if and only if the local independencies encoded by $\mathcal{G}$ hold in that distribution.* We refer to [Koller and Friedman, 2009] for a proof.

An example

An example of local independence and distribution factorization of Bayesian networks and Markov random fields is given in Figure 2.1.

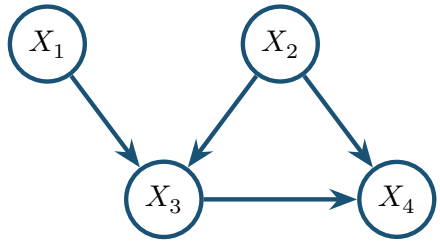
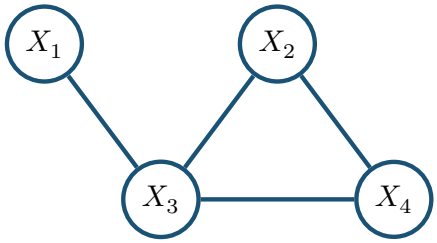
Bayesian network	Markov random field
	
$X_1 \perp\!\!\!\perp X_2$ $X_4 \perp\!\!\!\perp X_1 \mid X_2, X_3$	$X_1 \perp\!\!\!\perp X_2, X_4 \mid X_3$ $(X_2 \perp\!\!\!\perp X_1 \mid X_3, X_4; X_4 \perp\!\!\!\perp X_1 \mid X_2, X_3)$
$p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2)\times$ $\times\, p(x_3\|x_1, x_2)p(x_4\|x_2, x_3)$	$p(x_1, x_2, x_3, x_4) = \frac{1}{Z}\psi_1(x_1)\psi_2(x_2)\times$ $\times\, \psi_3(x_3)\psi_4(x_4)\psi_{13}(x_1, x_3)$ $\times\, \psi_{23}(x_2, x_3)\psi_{24}(x_2, x_4)\psi_{34}(x_3, x_4)$ $\times\, \psi_{234}(x_2, x_3, x_4)$

TABLE 2.1 An example of local independence and distribution factorization of Bayesian networks and Markov random fields. Conditional independencies displayed between parentheses are redundant and can be inferred from the others.

General observations

As we have seen, graphical models are useful because of their representation power. First, using a single graph structure, a graphical model can express all (conditional) independencies among a large set of random variables. Second, the same graphical model can also express the joint probability distribution over these random variables in a very compact form that is a factorization of local terms over smaller subsets of variables. We have also seen that these two points (*i.e.* conditional independence and distribution factorization) actually have a very tight connection.

2.1.2 Factor graphs

The factorization (2.7) may be cumbersome if one would like to model interactions over only a subset of cliques (instead of all cliques). For example, consider the following two joint distributions:

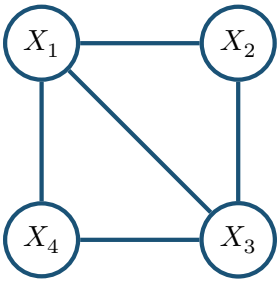
$$p(x_1, x_2, x_3, x_4) = \frac{1}{Z}\psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{14}(x_1, x_4)\psi_{23}(x_2, x_3)\psi_{34}(x_3, x_4), \quad (2.8)$$

$$p(x_1, x_2, x_3, x_4) = \frac{1}{Z}\psi_{123}(x_1, x_2, x_3)\psi_{134}(x_1, x_3, x_4). \quad (2.9)$$

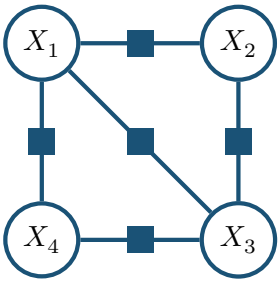
To model each of these distributions using graphical models, one needs the same graph in Figure 2.3a. However, according (2.7), the factorization of this graph is

$$p(x_1, x_2, x_3, x_4) = \frac{1}{Z}\psi_1(x_1)\psi_2(x_2)\psi_3(x_3)\psi_4(x_4)\psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\times \\ \times \psi_{14}(x_1, x_4)\psi_{23}(x_2, x_3)\psi_{34}(x_3, x_4)\psi_{123}(x_1, x_2, x_3)\psi_{134}(x_1, x_3, x_4). \quad (2.10)$$

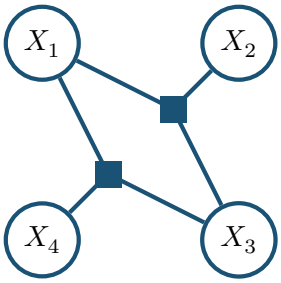
Therefore, one will have to explicitly set some local terms in the above to 1 to obtain (2.8) or (2.9).



(a) The factorization according to this graph may be cumbersome:
$p(\mathbf{x}) = \frac{1}{Z}\psi_1(x_1)\psi_2(x_2)\psi_3(x_3)\psi_4(x_4)\psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{14}(x_1, x_4)\times$
$\times \psi_{23}(x_2, x_3)\psi_{34}(x_3, x_4)\psi_{123}(x_1, x_2, x_3)\psi_{134}(x_1, x_3, x_4).$



(b) Factorization according to factors:
$p(\mathbf{x}) = \frac{1}{Z}\psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\times$
$\times \psi_{14}(x_1, x_4)\psi_{23}(x_2, x_3)\psi_{34}(x_3, x_4).$



(c) Factorization according to factors:
$p(\mathbf{x}) = \frac{1}{Z}\psi_{123}(x_1, x_2, x_3)\psi_{134}(x_1, x_3, x_4).$

FIGURE 2.3 Examples showing that factor graphs are more flexible. Suppose that we want to model only pairwise interactions as in (b) without factor graphs, then we need the graph shown in (a), which also contains cliques of sizes 1 and 3, thus we will need to explicitly set their corresponding potentials to 1. The same inconvenience applies if we want to model only triple-wise interactions as in (c).

To overcome this inconvenience, [Kschischang et al., 2001] introduced *factor graphs*. In addition to variable nodes (denoted by ovals: ○), a factor graph also contains *factor nodes* (denoted by squares: ■), each is connected to a subset of variable nodes to explicitly define a factor over those variables (this subset is thus called the *neighborhood* of that factor node). A factor graph encodes a family of distributions of the form

$$p(\mathbf{x}) = \frac{1}{Z}\prod_{F\in\mathcal{F}}\psi_F(\mathbf{x}_{\mathcal{N}(F)}) \quad \forall \mathbf{x}, \quad (2.11)$$

where $\mathcal{F}$ is the set of factor nodes, and $\mathcal{N}(F)$ denotes the neighborhood of F. Examples

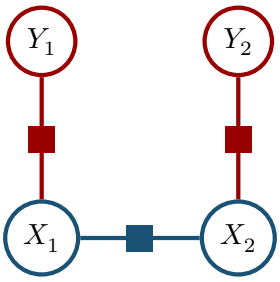


FIGURE 2.4 A conditional random field with observed variables Y_1, Y_2 and unobserved variables X_1, X_2. The factorization according to this factor graph is $p(\mathbf{x} \mid \mathbf{y}) = \frac{1}{Z}\psi_{11}(x_1; y_1)\psi_{22}(x_2; y_2)\psi_{12}(x_1, x_2)$.

of factor graphs for (2.8) and (2.9) are given in Figures 2.3b and 2.3c, respectively.

Conditional random fields

So far we have described an MRF as encoding a joint distribution. In practice, it is often the case that we can have access to some variables $\mathbf{Y}$ of the model, called *observed variables*. In this case, we are actually modeling a *conditional distribution* $p(\mathbf{X} \mid \mathbf{Y})$ and the corresponding MRF is generally called *conditional random field* (CRF). As for general MRFs, the factorization of CRFs can be easily expressed using factor graphs. An example is given in Figure 2.4.

2.2 MAP INFERENCE AND ENERGY MINIMIZATION

Finding the maximum a posteriori (MAP) configuration is a fundamental inference problem in undirected graphical models. This problem is described as follows.

Let $\mathbf{x} \in \mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$ denote an assignment to n discrete random variables $X_1, \ldots, X_n$ where each variable X_i takes values in a finite set of states (or *labels*) $\mathcal{X}_i$. Let $\mathcal{G}$ be a graph of n nodes with the set of cliques $\mathcal{C}$. Consider an MRF representing a joint distribution $p(\mathbf{X}) := p(X_1, \ldots, X_n)$ that factorizes according to $\mathcal{G}$, *i.e.* $p(\cdot)$ takes the form:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(\mathbf{x}_C) \quad \forall \mathbf{x} \in \mathcal{X}. \tag{2.12}$$

The MAP inference problem consists of finding the most likely assignment to the variables, *i.e.*:

$$\mathbf{x}^* \in \operatorname*{argmax}_{\mathbf{x} \in \mathcal{X}} p(\mathbf{x}) = \operatorname*{argmax}_{\mathbf{x} \in \mathcal{X}} \prod_{C \in \mathcal{C}} \psi_C(\mathbf{x}_C). \tag{2.13}$$

For each clique C, let $\mathcal{X}_C = \prod_{i \in C} \mathcal{X}_i$ be the set of its joint configurations and define

$$f_C(\mathbf{x}_C) = -\log \psi_C(\mathbf{x}_C) \quad \forall \mathbf{x}_C \in \mathcal{X}_C. \tag{2.14}$$

It is straightforward that the MAP inference problem (2.13) is equivalent to minimizing the following function, called the **energy** of the MRF:

$$e(\mathbf{x}) = \sum_{C \in \mathcal{C}} f_C(\mathbf{x}_C). \tag{2.15}$$

Therefore, MAP inference is also often referred to as *MRF energy minimization* or *MRF optimization* in the computer vision literature. This problem is known to be NP-hard in general [Shimony, 1994].

MAP inference has a wide range of applications in many fields. In particular, discrete MRFs have been ubiquitous in the computer vision field over the last decades thanks to their ability to model soft contextual constraints between random variables, which are extremely suitable for image or scene modeling in which usually involve interactions between a subset of pixels or scene components. We refer to [Wang et al., 2013] for a survey on MRF modeling, inference and learning in computer vision.

2.3 METHODS FOR MAP INFERENCE IN DISCRETE MRFS

MAP inference in discrete MRFs has been constantly attracting a significant amount of research over the last decades. Because of the NP-hardness, various approximate methods have been proposed and can be roughly grouped into four classes: *message passing*, *move making*, *combinatorial*, and *convex relaxation*. Later in Chapter 6 we will introduce a fifth class: *nonconvex relaxation*, which is one of the main contributions of the thesis.

Let us briefly review some of the most prominent methods in each class. We refer to [Kappes et al., 2015] for a recent comparative study of these methods on a wide variety of problems.

2.3.1 *Message passing methods*

The idea of message passing is to iteratively improve the labeling by passing local messages between neighboring nodes. The first algorithm of this class was proposed in [Pearl, 1982], called *belief propagation* (BP), for inference on Bayesian trees, in which the messages are the beliefs about the local configuration. The first generalization of BP is *loopy belief propagation* (LBP) [Frey and MacKay, 1997], which consists of BP in graphs with loops. LBP does not provide a guarantee on the convergence and on the quality of the solution. Recent generalizations of BP include *tree-reweighted message passing* (TRW) [Wainwright et al., 2005], which approximates the energy function based on a convex combination of trees and then maximizes a lower bound on the energy. However, the algorithm is not guaranteed to increase this bound and thus may not converge. Later, [Kolmogorov, 2006] developed a modification of this algorithm, called *sequential tree-reweighted message passing* (TRW-S), in which the lower bound is guaranteed not to decrease at each iteration, thus ensuring convergence.

2.3.2 *Move making methods*

These methods apply a sequence of minimizations over subsets of the label space, iteratively improving the current labeling. Each such minimization step is called a *move*. In move making methods, the energy is decreased after each move until convergence. These include graph-cut based methods such as α-*expansion* and $\alpha\beta$-*swap* [Boykov et al., 2001] for submodular, metric or semi-metric energy functions; *quadratic pseudo-boolean optimization* (QPBO) [Rother et al., 2007] for non-submodular energy functions. A nice generalization of α-expansion was proposed

in [Komodakis et al., 2008], called *fast primal-dual* (FastPD), which optimizes both the MRF optimization problem and its dual at each iteration, leading to a significant speed up. Recently, [Fix et al., 2014] generalized FastPD to higher-order MRFs.

2.3.3 Combinatorial methods

These methods view the problem as a combinatorial problem or an integer linear program and solve it exactly using combinatorial techniques such as branch-and-bound or branch-and-cut. They produce exact integer solutions but are usually intractable for large models. Recent work include [Kappes et al., 2011, Otten and Dechter, 2012, Hendrik Kappes et al., 2013, Savchynskyy et al., 2013, Kappes et al., 2016], *etc.*

2.3.4 Convex relaxation methods

These methods approximate the original labeling problem based on different relaxations and then use convex optimization techniques to solve the relaxed problem. The most popular class of these methods is *linear programming (LP) relaxation*. Some aforementioned work such as TRW [Wainwright et al., 2005], TRW-S [Kolmogorov, 2006] and FastPD [Komodakis et al., 2008] can also be considered to belong to this class, since they are based on LP relaxation.

An important line of work in this class is based on the *dual decomposition* framework (*c.f.* Section 4.1.1), first applied to MRFs by [Komodakis et al., 2011]. Dual decomposition consists of decomposing the original large and hard problem into a number of subproblems that are much easier to solve. In the original dual decomposition algorithm, [Komodakis et al., 2011] used *projected subgradient method* to update the dual objective at each iteration. Later, [Kappes et al., 2012] proposed to do that using *bundle method*, which was shown to perform better. Both methods are guaranteed to converge to the global optimum of the LP relaxation. However, they provide a very slow rate of convergence, namely $O(1/\epsilon^2)$ time complexity for an ϵ-accurate solution. This is mainly caused by the non-smoothness of the dual objective. Therefore, several authors proposed different smoothing based solutions to obtain better rates of convergence. These include [Martins et al., 2015, Jojic et al., 2010, Savchynskyy et al., 2011], *etc.*

Finally, other more complex convex relaxations have also been proposed, including the *quadratic programming relaxation* [Ravikumar and Lafferty, 2006] and *second order cone programming relaxation* [Waki et al., 2006]. These more sophisticated methods, however, provide worse approximations than the simple LP relaxation, as shown in [Kumar et al., 2009].

3

Graph and Hypergraph Matching

3.1 FEATURE CORRESPONDENCE AND GRAPH MATCHING

The task of finding correspondences between two sets of features is a fundamental problem and has a wide range of applications in computer vision and pattern recognition.

Let $\mathcal{V}_1$ and $\mathcal{V}_2$ be two sets of feature points. The correspondence (or matching) between them can then be represented by a matrix, called *assignment matrix*, $\mathbf{X} \in \{0,1\}^{|\mathcal{V}_1|\times|\mathcal{V}_2|}$ with elements $(X_{i_1 i_2})_{i_1\in\mathcal{V}1, i_2\in\mathcal{V}_2}$, where each row corresponds to a point in $\mathcal{V}_1$ and each column corresponds to a point in $\mathcal{V}_2$. If a point $i_1 \in \mathcal{V}_1$ is a correspondence of a point $i_2 \in \mathcal{V}_2$ (or alternatively we can say i_1 and i_2 are matched) then $X_{i_1 i_2} = 1$, otherwise $X_{i_1 i_2} = 0$. For example, the following matrix

$$\mathbf{X} = \begin{array}{c} \begin{array}{ccccc} i_2 & j_2 & k_2 & l_2 & m_2 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{array}{l} i_1 \\ j_1 \\ k_1 \\ l_1 \end{array} \tag{3.1}$$

represents the following correspondences: $i_1 \leftrightarrow i_2, j_1 \leftrightarrow k_2, k_1 \leftrightarrow j_2$ and $l_1 \leftrightarrow m_2$.

Depending on the application, the assignment matrix can obey different constraints. For example, if one feature point can only have exactly one correspondence (in this case $|\mathcal{V}_1|$ and $|\mathcal{V}_2|$ is supposed to be equal), then the sum of each row or each column of $\mathbf{X}$ must be 1, and we call this *one-to-one matching*. If any point can have one or no correspondence (this is the case for example when $|\mathcal{V}_1| \neq |\mathcal{V}_2|$), then the sum of each row or each column of $\mathbf{X}$ must be less than or equal to 1, and we call this *one-to-(at most)-one matching*. If one point can have multiple correspondences then we do not need any constraints on the row or the column of $\mathbf{X}$, and we call this *one-to-many* matching. Obviously one can model any kind of similar configurations (*e.g.* mixtures of the above). A configuration that is often used in practice — when the numbers of feature points are different, *e.g.* $|\mathcal{V}_1| < |\mathcal{V}_2|$, and one wants to obtain the maximum number of one-to-one correspondences — is to have the sum of each row equal to 1 and the sum of each column less than or equal to 1, *i.e.* $\mathbf{X} \in \overline{\mathcal{M}}$, defined by

$$\overline{\mathcal{M}} = \left\{ \mathbf{X} \in \{0,1\}^{|\mathcal{V}_1|\times|\mathcal{V}_2|} \;\middle|\; \begin{array}{l} \sum_{i_2\in\mathcal{V}_2} X_{i_1 i_2} = 1 \;\forall i_1 \in \mathcal{V}_1 \\ \sum_{i_1\in\mathcal{V}_1} X_{i_1 i_2} \leq 1 \;\forall i_2 \in \mathcal{V}_2 \end{array} \right\}. \tag{3.2}$$

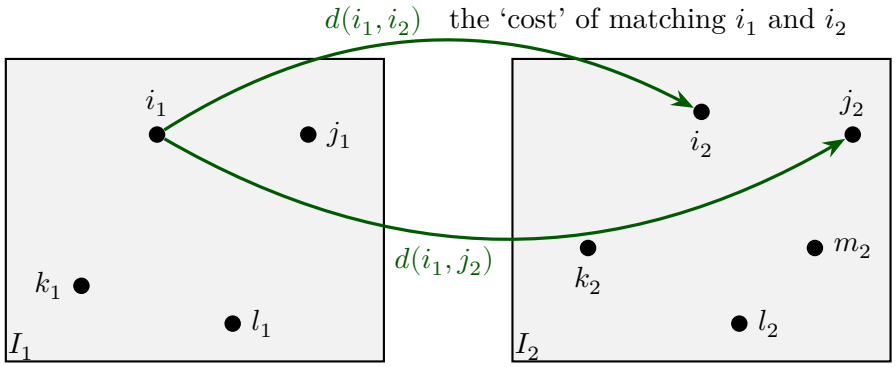


FIGURE 3.1 An illustration of finding correspondences between two sets of features. This task can be solved by minimizing the total dissimilarity between all matched points. One can think of paying some 'cost' $d(i_1, i_2)$ whenever matching a pair of features (i_1, i_2). This cost usually represents the dissimilarity between i_1 and i_2, *e.g.* difference in colors. Finding one-to-one correspondence can then be reduced to solving a linear assignment problem.

This is also called one-to-one matching. Without loss of generality, in the sequel we suppose $\mathbf{X} \in \overline{\mathcal{M}}$ and will refer to this as the one-to-one constraint.

Suppose that for each pair of feature points (i_1, i_2) we can compute some dissimilarity measure $d(i_1, i_2)$ between them (based on their local characteristics for example). Naturally, one may think about finding (one-to-one) correspondences as maximizing the total similarity between the matched points, *i.e.* solving

$$\min \sum_{i_1 \in \mathcal{V}_1} \sum_{i_2 \in \mathcal{V}_2} d(i_1, i_2) X_{i_1 i_2} \quad \text{s.t. } \mathbf{X} \in \overline{\mathcal{M}}. \tag{3.3}$$

The dissimilarity measure $d(i_1, i_2)$ can be seen as the "cost" of matching i_1 and i_2, and the problem becomes minimizing the total matching cost (*c.f.* Figure (3.1)). This problem is known as the *linear assignment problem* (LAP), and can be solved exactly in polynomial time using *e.g.* the Hungarian algorithm [Kuhn, 1955] and its variants.

A major issue with the formulation (3.3) is that it does not take into account any possible structural information or spatial relationship between the features. This kind of information is in fact extremely useful when the dissimilarity measure in (3.3) is not reliable enough (which is the case *e.g.* when the local characteristics of the feature points are not discriminative enough and contain ambiguities). A solution to this is *graph matching*. The general idea is to associate each set of features an attributed graph, where the node attributes describe local characteristics, while the edge attributes describe structural relationships. Then, the matching task seeks to minimize an objective function that contains not only the dissimilarity between nodes, but also between edges. Suppose that the corresponding graphs are $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ and for any pair of edges $i_1 j_1 \in \mathcal{E}_1, i_2 j_2 \in \mathcal{E}_2$ we can compute a dissimilarity measure $d(i_1 j_1, i_2 j_2)$ (*e.g.* difference in lengths or angles), then graph matching consists of solving the following optimization problem:

$$\begin{aligned} \min \quad & \sum_{i_1 \in \mathcal{V}_1} \sum_{i_2 \in \mathcal{V}_2} d(i_1, i_2) X_{i_1 i_2} + \sum_{i_1 j_1 \in \mathcal{E}_1} \sum_{i_2 j_2 \in \mathcal{E}_2} d(i_1 j_1, i_2 j_2) X_{i_1 i_2} X_{j_1 j_2}, \\ \text{s.t.} \quad & \mathbf{X} \in \overline{\mathcal{M}}. \end{aligned} \tag{3.4}$$

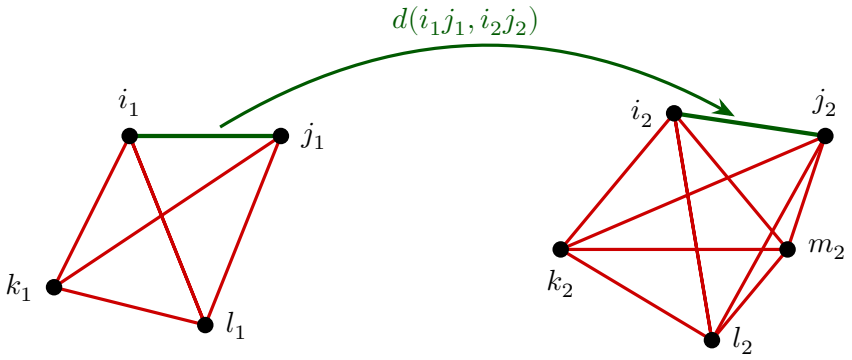


FIGURE 3.2 An illustration of graph-based feature correspondence. Each set of features is associated to a graph that represents the geometric or structural relationships between them. The objective cost now contains not only the dissimilarity between graph nodes, but also between graph edges.

This problem can be seen as the *quadratic assignment problem* (QAP) in general form, known as Lawler's QAP [Lawler, 1963], which is NP-complete [Sahni and Gonzalez, 1976, Burkard et al., 1998]. In the computer vision literature, the objective in (3.4) is called the *energy function*, and the dissimilarities $d(i_1, j_2)$ and $d(i_1j_1, i_2j_2)$ are called *unary* and *pairwise potentials*, respectively.

A straightforward extension of the above approach is to use *hypergraphs* instead of regular graphs. An edge in a hypergraph, also called a *hyperedge*, may contain more than two nodes. For example, if we use hypergraphs of degree 3, *i.e.* each edge contains at most 3 nodes, then similarly to the previous cases, the matching problem can be formulated as:

$$\begin{aligned} \min \quad & \{\text{unary}\} + \{\text{pairwise}\} + \sum_{i_1j_1k_1 \in \mathcal{E}_1} \sum_{i_2j_2k_2 \in \mathcal{E}_2} d(i_1j_1k_1, i_2j_2k_2) X_{i_1i_2} X_{j_1j_2} X_{k_1k_2}, \\ \text{s.t.} \quad & \mathbf{X} \in \overline{\mathcal{M}}, \end{aligned} \tag{3.5}$$

where $\{\text{unary}\} + \{\text{pairwise}\}$ is the same as in (3.4). The last sum in the above formulation is usually called the *higher-order terms* (and more specifically in this case, the *third-order* ones). As such, hypergraph matching is also often referred to as *higher-order matching*, whereas regular graph matching is called *pairwise matching*. In this thesis, we often refer to both as simply *graph matching* and when necessary, we specify it to be *pairwise* or *higher-order*.

Clearly, higher-order matching models have more expressive power than pairwise models, and thus they can better incorporate structural information to achieve more accurate matching results. We refer to [Duchenne et al., 2011] for a more detailed discussion on this matter.

3.2 LINEAR ALGEBRA REFORMULATIONS

For further convenience, let us reformulate graph matching using linear algebra notations. These formulations are widely used in the computer vision literature.

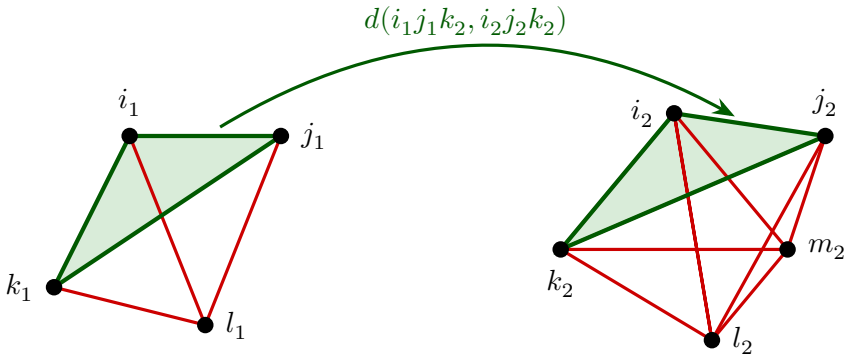


FIGURE 3.3 An illustration of hypergraph-based feature correspondence. In addition to graph nodes and edges, one can go a step further and include the dissimilarity between subsets of nodes, called *hyperedges*, into the cost function. Hyperedges can better incorporate structural information than regular edges.

3.2.1 *Review of tensors*

A real-valued D^{th}-order tensor $\mathbf{F}$ is a multidimensional array belonging to $\mathbb{R}^{n_1 \times n_2 \times \cdots \times n_D}$ (where $n_1, n_2, \dots, n_D$ are positive integers). Each dimension of $\mathbf{F}$ is called a *mode*. The elements of $\mathbf{F}$ are denoted by $F_{i_1 i_2 \dots i_D}$ where i_d is the index along the mode d.

We call the *multilinear form* associated to a tensor $\mathbf{F}$ a function $F : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \times \cdots \times \mathbb{R}^{n_D} \to \mathbb{R}$ defined by

$$F(\mathbf{x}_1, \dots, \mathbf{x}_D) = \sum_{i_1=1}^{n_1} \cdots \sum_{i_D=1}^{n_D} F_{i_1 i_2 \dots i_D} x_{i_1}^1 x_{i_2}^2 \cdots x_{i_D}^D, \tag{3.6}$$

where $\mathbf{x}_d = (x_1^d, x_2^d, \dots, x_{n_d}^d) \in \mathbb{R}^{n_d}$ for $d = 1, 2, \dots, D$.

A tensor can be multiplied by a vector at a specific mode. Let $\mathbf{v} = (v_1, v_2, \dots, v_{n_d})$ be an n_d dimensional vector. The *mode-d product* of $\mathbf{F}$ and $\mathbf{v}$, denoted $\mathbf{F} \bigotimes_d \mathbf{v}$, is a $(D-1)^{\text{th}}$-order tensor $\mathbf{G}$ of dimensions $n_1 \times \cdots \times n_{d-1} \times n_{d+1} \times \cdots \times n_D$ defined by

$$G_{i_1 \dots i_{d-1} i_{d+1} \dots i_D} = \sum_{i_d=1}^{n_d} F_{i_1 \dots i_d \dots i_D} v_{i_d} \quad \forall i_{[1,D] \setminus d}. \tag{3.7}$$

The multiplication is only valid if $\mathbf{v}$ has the same dimension as the mode d of $\mathbf{F}$.

The product of a tensor and multiple vectors (at multiple modes) is defined as the consecutive product of the tensor and each vector (at the corresponding mode). The order of the multiplied vectors does not matter. For example, the product of a 4^{th}-order tensor $\mathbf{F} \in \mathbb{R}^{n_1 \times n_2 \times n_3 \times n_4}$ and two vectors $\mathbf{u} \in \mathbb{R}^{n_2}, \mathbf{v} \in \mathbb{R}^{n_4}$ at the modes 2 and 4 (respectively) is an $n_1 \times n_3$ tensor $\mathbf{G} = \mathbf{F} \bigotimes_2 \mathbf{u} \bigotimes_4 \mathbf{v} = \mathbf{F} \bigotimes_4 \mathbf{v} \bigotimes_2 \mathbf{u}$, where

$$G_{i_1 i_3} = \sum_{i_2=1}^{n_2} \sum_{i_4=1}^{n_4} F_{i_1 i_2 i_3 i_4} u_{i_2} v_{i_4} \quad \forall i_1, i_3. \tag{3.8}$$

Let us consider for convenience the notation $\mathbf{F} \bigotimes_{\mathcal{I}} \mathcal{M}$ to denote the product of $\mathbf{F}$ with the set of vectors $\mathcal{M}$, at the modes specified by the set of indices $\mathcal{I}$ with $|\mathcal{I}| = |\mathcal{M}|$.

Since the order of the vectors and the modes must agree, $\mathcal{M}$ and $\mathcal{I}$ are supposed to be ordered sets. By convention, $\mathbf{F} \bigotimes_{\mathcal{I}} \mathcal{M} = \mathbf{F}$ if $\mathcal{M} = \emptyset$. Using this notation, the product in the previous example becomes

$$\mathbf{G} = \mathbf{F} \bigotimes_{\{2,4\}} \{\mathbf{u}, \mathbf{v}\} = \mathbf{F} \bigotimes_{\{4,2\}} \{\mathbf{v}, \mathbf{u}\}. \tag{3.9}$$

Let us also consider the notation $\bigotimes_{d=a}^{b}$ to denote a sequence of products from mode a to mode b:

$$\mathbf{F} \bigotimes_{d=a}^{b} \mathbf{x}_d = \mathcal{F} \otimes_a \mathbf{x}_a \otimes_{a+1} \mathbf{x}_{a+1} \cdots \otimes_b \mathbf{x}_b. \tag{3.10}$$

By convention, $\mathbf{F} \bigotimes_{d=a}^{b} \mathbf{x}_d = \mathbf{F}$ if $a > b$. Using this notation, it is straightforward to see that the multilinear form (3.6) can be re-written as

$$F(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_D) = \mathbf{F} \otimes_1 \mathbf{x}_1 \otimes_2 \mathbf{x}_2 \cdots \otimes_D \mathbf{x}_D = \mathbf{F} \bigotimes_{d=1}^{D} \mathbf{x}_d. \tag{3.11}$$

3.2.2 *Reformulation of graph matching*

Let $n_1 = |\mathcal{V}_1|, n_2 = |\mathcal{V}_2|$ and $n = n_1 n_2$. For an $n_1 \times n_2$ matrix $\mathbf{V}$, let $\text{vec}(\mathbf{V}) \in \mathbb{R}^n$ denote its row-wise vectorized replica; and inversely for a vector $\mathbf{v} \in \mathbb{R}^n$ let $\text{mat}(\mathbf{v})$ denote its corresponding $n_1 \times n_2$ reshaped matrix. We should note that while reshaping a vector to a matrix requires specifying the dimensions, we use $\text{mat}(\cdot)$ exclusively for $n_1 \times n_2$ matrices.

Now for an assignment matrix $\mathbf{X} \in \{0,1\}^{n_1 \times n_2}$, let $\mathbf{x} = \text{vec}(\mathbf{X}) \in \{0,1\}^n$, called the assignment vector. Clearly, each assignment $i_1 \leftrightarrow i_2$ corresponds to an index i in $\mathbf{x}$, where $1 \le i \le n$.

Consider the 3$^{\text{rd}}$-order potentials $d(i_1 j_1 k_1, i_2 j_2 k_2)$ in (3.5). It is straightforward that these terms can be represented by a 3$^{\text{rd}}$-order tensor $\mathbf{F}^3$ of dimensions $n \times n \times n$ with elements $F_{ijk} = d(i_1 j_1 k_1, i_2 j_2 k_2)$ where i, j, k correspond to the assignments $i_1 \leftrightarrow i_2, j_1 \leftrightarrow j_2, k_1 \leftrightarrow k_2$, respectively. Therefore, the 3$^{\text{rd}}$-order terms in (3.5) can be expressed as

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} F_{ijk} x_i x_j x_k. \tag{3.12}$$

We should note that some elements of $\mathbf{F}^3$ must be set to zeros if they do not appear in (3.5) (*e.g.* F_{iik} or F_{iki}). Clearly, according to (3.6) and (3.11) we have

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} F_{ijk} x_i x_j x_k = F^3(\mathbf{x}, \mathbf{x}, \mathbf{x}) = \mathbf{F}^3 \otimes_1 \mathbf{x} \otimes_2 \mathbf{x} \otimes_3 \mathbf{x}, \tag{3.13}$$

where $F^3(\cdot, \cdot, \cdot)$ is the multilinear form of the tensor $\mathbf{F}^3$.

More generally, the d^{th}-order potentials can be represented by a d^{th}-order tensor

$\mathbf{F}^d$. Therefore, the 3rd (or higher) order graph matching can be reformulated as

$$\begin{aligned} \min \quad & E(\mathbf{x}) := \mathbf{F}^1 \otimes_1 \mathbf{x} + \mathbf{F}^2 \otimes_1 \mathbf{x} \otimes_2 \mathbf{x} + \mathbf{F}^3 \otimes_1 \mathbf{x} \otimes_2 \mathbf{x} \otimes_3 \mathbf{x} + \dots, \\ \text{s.t.} \quad & \text{mat}(\mathbf{x}) \in \overline{\mathcal{M}}, \end{aligned} \tag{3.14}$$

where the dots contain possible potential terms of higher degrees.

For the special case of pairwise matching, tensor is not needed and so we can get a more familiar expression. Indeed, since $\mathbf{F}^1$ is a vector and $\mathbf{F}^2$ is a matrix, for pairwise matching the energy in (3.14) becomes

$$E(\mathbf{x}) = \mathbf{F}^1 \cdot \mathbf{x} + \mathbf{F}^2 \mathbf{x} \cdot \mathbf{x}. \tag{3.15}$$

For more convenience let us denote $\mathbf{u} := \mathbf{F}^1$ to be the unary potential vector and $\mathbf{P} := \mathbf{F}^2$ to be the pairwise potential matrix. Pairwise graph matching can then be reformulated as

$$\begin{aligned} \min \quad & \mathbf{u}^\top \mathbf{x} + \mathbf{x}^\top \mathbf{P} \mathbf{x}, \\ \text{s.t.} \quad & \text{mat}(\mathbf{x}) \in \overline{\mathcal{M}}, \end{aligned} \tag{3.16}$$

Since $\mathbf{x}$ is binary, we have $u_i x_i = u_i x_i x_i \ \forall i$ and thus we obtain the following alternative formulation:

$$\begin{aligned} \min \quad & \mathbf{x}^\top \mathbf{M} \mathbf{x}, \\ \text{s.t.} \quad & \text{mat}(\mathbf{x}) \in \overline{\mathcal{M}}, \end{aligned} \tag{3.17}$$

where

$$\mathbf{M} = \mathbf{P} + \text{diag}(\mathbf{u}). \tag{3.18}$$

The potential matrix $\mathbf{M}$ represents the dissimilarity between the correspondences. Equivalently one can choose $\mathbf{M}$ to represent the *similarity*, and replace the above minimization by maximization, then $\mathbf{M}$ is called the *affinity matrix* and the objective function is called the *score*.

3.3 METHODS FOR GRAPH AND HYPERGRAPH MATCHING

Graph matching has been an active research topic in the computer vision field for the past decades. In this section, we review the most prominent methods in recent literature.

In the recent literature, [Gold and Rangarajan, 1996] proposed a graduated assignment algorithm to iteratively solve a series of convex approximations to the matching problem. In [Leordeanu and Hebert, 2005], a spectral matching based on the rank-1 approximation of the affinity matrix was introduced, which was later improved in [Cour et al., 2007] by incorporating affine constraints towards a tighter relaxation. In [Leordeanu et al., 2009], an integer projected fixed point algorithm that solves a sequence of first-order Taylor approximations using Hungarian method [Kuhn, 1955] was proposed, while in [Torresani et al., 2013] the dual of the matching problem was considered to obtain a lower-bound on the energy, via dual decomposition. In [Cho et al., 2010], a reweighted random walk variant was used to address graph matching, while a convex-concave relaxation was proposed in [Zhou and De la Torre, 2012] based on the factorization of the affinity matrix into smaller ones. Their inspiration was the path-following approach [Zaslavskiy et al., 2009] exploiting a more restricted formulation,

known as Koopmans-Beckmann's QAP [Koopmans and Beckmann, 1957]. More recently, [Cho et al., 2014] proposed a max-pooling strategy within the graph matching framework that is very robust to outliers.

Over the last few years, researchers have proposed higher-order graph matching models to better incorporate structural similarities and achieve more accurate results than pairwise matching [Zass and Shashua, 2008, Duchenne et al., 2011]. For solving such high-order models, [Zass and Shashua, 2008] viewed the matching problem as a probabilistic model that is solved using an iterative successive projection algorithm. The extension of pairwise methods to deal with higher-order potentials was also considered like for example in [Duchenne et al., 2011] through a tensor matching (extended from [Leordeanu and Hebert, 2005]), or in [Zeng et al., 2010] through a third-order dual decomposition algorithm (originating from [Torresani et al., 2013]), or in [Lee et al., 2011] through a high-order reweighted random walk matching (extension of [Cho et al., 2010]). Recently, [Nguyen et al., 2015] developed a block coordinate ascent algorithm for solving third-order graph matching. They lifted the third-order problem to a fourth-order one which, after a convexification step, is solved by a sequence of linear or quadratic assignment problems.

4

Alternating Direction Method of Multipliers

In this chapter, we review the alternating direction method of multipliers (ADMM). We start by introducing the classical ADMM, which is for solving convex, separable, two-block problems. Then we give a presentation of different generalizations and extensions of ADMM.

4.1 CLASSICAL ALTERNATING DIRECTION METHOD OF MULTIPLIERS

4.1.1 *Motivation and algorithm*

Consider the following convex optimization problem:

$$\begin{aligned} \min \quad & f(\mathbf{x}) + g(\mathbf{z}) \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} = \mathbf{c}, \end{aligned} \tag{4.1}$$

with variables $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{z} \in \mathbb{R}^m$, where $\mathbf{A} \in \mathbb{R}^{p\times n}, \mathbf{B} \in \mathbb{R}^{p\times m}, \mathbf{c} \in \mathbb{R}^p$ and $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}, g : \mathbb{R}^m \to \mathbb{R} \cup \{+\infty\}$ are (extended) real-valued **convex** functions.

The Lagrangian of (4.1) is defined by

$$L(\mathbf{x}, \mathbf{z}, \mathbf{y}) = f(\mathbf{x}) + g(\mathbf{z}) + \mathbf{y}^\top(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}), \tag{4.2}$$

and the dual function is

$$h(\mathbf{y}) = \min_{\mathbf{x}\in\mathbb{R}^n, \mathbf{z}\in\mathbb{R}^m} L(\mathbf{x}, \mathbf{z}, \mathbf{y}), \tag{4.3}$$

where $\mathbf{y} \in \mathbb{R}^p$ is called the dual variable (or alternatively the Lagrangian multiplier), $\mathbf{x}$ and $\mathbf{z}$ are called the primal variables. The corresponding so-called dual problem is

$$\max_{\mathbf{y}\in\mathbb{R}^p} h(\mathbf{y}). \tag{4.4}$$

Since our problem (4.1) is convex, duality theory[1] tells us that the maximum value of (4.4) and the minimum value of (4.1) coincide (under the trivial assumption that there exist $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{z} \in \mathbb{R}^m$ such that $\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} = \mathbf{c}$, *i.e.* problem (4.1) is feasible). In addition, if $\mathbf{y}^*$ is an optimal solution to (4.4) then an optimal solution to (4.1) can be recovered by

$$(\mathbf{x}^*, \mathbf{z}^*) = \operatorname*{argmin}_{(\mathbf{x},\mathbf{z})\in\mathbb{R}^n\times\mathbb{R}^m} L(\mathbf{x}, \mathbf{z}, \mathbf{y}^*). \tag{4.5}$$

[1] See *e.g.* Chapter 5 of [Boyd and Vandenberghe, 2004].

Therefore, one natural way to solve (4.1) is to solve (4.4) instead. Since the latter is an unconstrained optimization problem, it can be solved using *e.g.* the usual *gradient ascent* method if h is differentiable, which consists of updating $\mathbf{y} \leftarrow \mathbf{y} + \alpha \nabla h(\mathbf{y})$. It is straightforward to prove that if $(\mathbf{x}(\mathbf{y}), \mathbf{z}(\mathbf{y}))$ — here we view $\mathbf{x}$ and $\mathbf{z}$ as functions of $\mathbf{y}$ — is an optimal solution to (4.3) then $\nabla h(\mathbf{y}) = \mathbf{A}\mathbf{x}(\mathbf{y}) + \mathbf{B}\mathbf{z}(\mathbf{y}) - \mathbf{c}$. Therefore, we can deduce that the gradient ascent algorithm for solving (4.4) consists of the following updates:

$$\left(\mathbf{x}^{(k+1)}, \mathbf{z}^{(k+1)}\right) \leftarrow \underset{(\mathbf{x},\mathbf{z}) \in \mathbb{R}^n \times \mathbb{R}^m}{\operatorname{argmin}} L(\mathbf{x}, \mathbf{z}, \mathbf{y}^{(k)}), \tag{4.6}$$

$$\mathbf{y}^{(k+1)} \leftarrow \mathbf{y}^{(k)} + \alpha^{(k)} \left(\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{z}^{(k+1)} - \mathbf{c}\right), \tag{4.7}$$

where k is the iteration counter and $\alpha^{(k)} > 0$ is the step-size. If h is non-differentiable, then the quantity $\mathbf{A}\mathbf{x}(\mathbf{y}) + \mathbf{B}\mathbf{z}(\mathbf{y}) - \mathbf{c}$ now becomes the negative of a *subgradient*[2] of $-h$ at $\mathbf{y}$, so the above iterates are still valid for solving (4.4) and are known as the *subgradient method* [Shor et al., 1985] (though we should note that unlike in gradient ascent, the update (4.7) may not increase the value of h at each iteration; however, the algorithm is still guaranteed to converge to an optimal solution with suitable choices of $\alpha^{(k)}$).

We have described a technique to reduce the original problem (4.1) to solving its dual problem (4.4) via subgradient method (or via gradient ascent if the dual function is differentiable). This technique is known as the *dual subgradient method* (respectively *dual ascent method*). Now consider the update step (4.6). From (4.2) it is clear that the optimization problem in (4.6) can be re-written as

$$\min_{\mathbf{x} \in \mathbb{R}^n} \left\{ f(\mathbf{x}) + \langle \mathbf{y}^{(k)}, \mathbf{A}\mathbf{x} \rangle \right\} + \min_{\mathbf{z} \in \mathbb{R}^m} \left\{ g(\mathbf{z}) + \langle \mathbf{y}^{(k)}, \mathbf{B}\mathbf{z} \rangle \right\}, \tag{4.8}$$

which can be solved separately (and in parallel) with respect to $\mathbf{x}$ and $\mathbf{z}$. We have thus reduced (4.1) to solving a series of smaller and independent subproblems as in (4.8). This is clearly possible only if the objective function in (4.1) is separable. In this case, we refer to the dual subgradient (or dual ascent) method as *dual decomposition*.

A major drawback of the above approach for solving (4.1) is that the dual function is often not differentiable, and thus one has to use dual subgradient instead of dual ascent, which results in very slow convergence. Indeed, to achieve an ϵ-approximate solution, dual subgradient requires $\mathcal{O}(1/\epsilon^2)$ iterations while dual ascent requires only $\mathcal{O}(1/\epsilon)$ iterations [Shor et al., 1985, Nesterov, 2013]. A solution to overcome this is to view (4.1) in the following equivalent form:

$$\begin{aligned} \min \quad & f(\mathbf{x}) + g(\mathbf{z}) + \frac{\rho}{2} \|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}\|_2^2, \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} = \mathbf{c}, \end{aligned} \tag{4.9}$$

where $\rho > 0$ is called the *penalty parameter*. The Lagrangian of this problem is

$$L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{y}) = f(\mathbf{x}) + g(\mathbf{z}) + \mathbf{y}^\top (\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}) + \frac{\rho}{2} \|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}\|_2^2, \tag{4.10}$$

[2] A vector $\mathbf{s}$ is called a subgradient of a function h at $\mathbf{u}$ if and only if $h(\mathbf{v}) \geq h(\mathbf{u}) + \mathbf{s}^\top(\mathbf{v} - \mathbf{u})$ $\forall \mathbf{v}$. We write $\mathbf{s} \in \partial h(\mathbf{u})$, where $\partial h(\mathbf{u})$ is the *subdifferential* of h at $\mathbf{u}$, *i.e.* the set of all subgradients of h at $\mathbf{u}$. If h is convex and differentiable then $\partial h = \{\nabla h\}$.

which is also called the *augmented* Lagrangian of (4.1). The associated dual function now becomes

$$h_\rho(\mathbf{y}) = \min_{\mathbf{x}\in\mathbb{R}^n, \mathbf{z}\in\mathbb{R}^m} L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{y}). \tag{4.11}$$

The benefit of adding a penalty term is that $h_\rho(\mathbf{y})$ can be shown to be differentiable under mild conditions [Boyd et al., 2011]. We can now apply dual ascent to solve (4.9) (and thus (4.1)), where we use ρ as a constant step-size:

$$\left(\mathbf{x}^{(k+1)}, \mathbf{z}^{(k+1)}\right) \leftarrow \operatorname*{argmin}_{(\mathbf{x},\mathbf{z})\in\mathbb{R}^n\times\mathbb{R}^m} L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{y}^{(k)}), \tag{4.12}$$

$$\mathbf{y}^{(k+1)} \leftarrow \mathbf{y}^{(k)} + \rho\left(\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{z}^{(k+1)} - \mathbf{c}\right). \tag{4.13}$$

This is known as the *method of multipliers* for solving (4.1).

Now let us have a closer look at the update step (4.12). As the objective function in (4.9) is not separable (unlike (4.1)), this step cannot be decomposed into smaller subproblems like (4.8). This is a serious issue because solving (4.12) can be very expensive in practice for large-scale problems. Fortunately, it turns out that for (4.12), instead of minimizing jointly with respect to $\mathbf{x}$ and $\mathbf{z}$, one can do that alternatively over each variable:

$$\mathbf{x}^{(k+1)} \leftarrow \operatorname*{argmin}_{\mathbf{x}} L_\rho(\mathbf{x}, \mathbf{z}^{(k)}, \mathbf{y}^{(k)}), \tag{4.14}$$

$$\mathbf{z}^{(k+1)} \leftarrow \operatorname*{argmin}_{\mathbf{z}} L_\rho(\mathbf{x}^{(k+1)}, \mathbf{z}, \mathbf{y}^{(k)}), \tag{4.15}$$

and yet the algorithm is still guaranteed to converge, and at a good rate. This is called the *alternating direction method of multipliers* (ADMM), presented in a more complete form in Algorithm 4.1.

ALGORITHM 4.1 ADMM for solving two-block problems.

1: Initialization: $k \leftarrow 0$, $\mathbf{x}^{(0)} \in \mathbb{R}^n, \mathbf{z}^{(0)} \in \mathbb{R}^m$ and $\mathbf{y}^{(0)} \in \mathbb{R}^p$.

2: Update $\mathbf{x}$:

$$\mathbf{x}^{(k+1)} \leftarrow \operatorname*{argmin}_{\mathbf{x}} L_\rho(\mathbf{x}, \mathbf{z}^{(k)}, \mathbf{y}^{(k)}). \tag{4.16}$$

3: Update $\mathbf{z}$:

$$\mathbf{z}^{(k+1)} \leftarrow \operatorname*{argmin}_{\mathbf{z}} L_\rho(\mathbf{x}^{(k+1)}, \mathbf{z}, \mathbf{y}^{(k)}). \tag{4.17}$$

4: Update $\mathbf{y}$:

$$\mathbf{y}^{(k+1)} \leftarrow \mathbf{y}^{(k)} + \rho(\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{z}^{(k+1)} - \mathbf{c}). \tag{4.18}$$

Let $k \leftarrow k+1$ and go to Step 2.

Originally proposed by [Glowinski and Marroco, 1975] and [Gabay and Mercier, 1975], ADMM has recently attracted a lot of attention from the machine learning and computer vision fields thanks to its excellent performance and flexibility, especially on large-scale problems. We refer to [Boyd et al., 2011] for historical notes and references on ADMM, as well as for a review on ADMM for distributed optimization and statistical learning.

ADMM for variables under set constraints

In the previous formulation, the functions f and g are allowed to be *non-differentiable* and to take the $+\infty$ value, so that (4.1) can also cover problems with set constraints, which is often the case in practice. Indeed, consider the following problem:

$$\begin{aligned} \min \quad & f(\mathbf{x}) + g(\mathbf{z}) \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} = \mathbf{c}, \\ & \mathbf{x} \in \mathcal{X}, \mathbf{z} \in \mathcal{Z}, \end{aligned} \tag{4.19}$$

where $\mathcal{X} \subseteq \mathbb{R}^n, \mathcal{Z} \subseteq \mathbb{R}^m$ are closed convex sets.

Let $\delta_{\mathcal{S}}(\cdot)$ denote the indicator function of the set $\mathcal{S}$, *i.e.*

$$\delta_{\mathcal{S}}(s) = \begin{cases} 0 & \text{if } s \in \mathcal{S}, \\ +\infty & \text{otherwise.} \end{cases} \tag{4.20}$$

Denote $f_0(\mathbf{x}) := f(\mathbf{x}) + \delta_{\mathcal{X}}(\mathbf{x})$ and $g_0(\mathbf{z}) := g(\mathbf{z}) + \delta_{\mathcal{Z}}(\mathbf{z})$, then clearly (4.19) is reduced to (4.1) with objective function $f_0(\mathbf{x}) + g_0(\mathbf{z})$.

4.1.2 Convergence

In this section we present some basic, yet very general, convergence results of ADMM (Algorithm 4.1) for solving (4.1). In [Boyd et al., 2011], ADMM was shown to converge under the following two assumptions.

Assumption 4.1. *The functions f and g are closed, proper, and convex.*

Assumption 4.2. *The unaugmented Lagrangian L_0 has a saddle point.*

The second assumption means that there exist $(\mathbf{x}^*, \mathbf{z}^*, \mathbf{y}^*)$ such that

$$L_0(\mathbf{x}^*, \mathbf{z}^*, \mathbf{y}) \le L_0(\mathbf{x}^*, \mathbf{z}^*, \mathbf{y}^*) \le L_0(\mathbf{x}, \mathbf{z}, \mathbf{y}^*). \tag{4.21}$$

It is straightforward to see that (4.21) is equivalent — given that f and g are convex — to the following so-called Karush–Kuhn–Tucker (KKT) conditions for problem (4.1):

$$\mathbf{c} = \mathbf{A}\mathbf{x}^* + \mathbf{B}\mathbf{z}^*, \tag{4.22}$$

$$\mathbf{0} \in \partial f(\mathbf{x}^*) + \mathbf{A}^\top \mathbf{y}^*, \tag{4.23}$$

$$\mathbf{0} \in \partial g(\mathbf{z}^*) + \mathbf{B}^\top \mathbf{y}^*, \tag{4.24}$$

where we recall that $\partial h(\mathbf{u})$ denotes the subdifferential of a function h at $\mathbf{u}$. Note that since (4.1) is convex, the above KKT conditions are *necessary and sufficient* for $(\mathbf{x}^*, \mathbf{z}^*, \mathbf{y}^*)$ to be a primal-dual optimum. Therefore, Assumption 4.2 basically means that problem (4.1) has a primal-dual optimal solution.

Regarding Assumption 4.1, [Boyd et al., 2011] stated that it ensures the *solvability*[3] for the subproblems (4.16) and (4.17) at every iteration. This is not correct,

[3] An optimization problem is said to be *solvable* if it is feasible, bounded below and its optimal value can be attained. As an example, the convex program $\min\{\frac{1}{x} \mid x \ge 1\}$ is feasible and bounded below, but not solvable.

however, as pointed out by [Chen et al., 2017]. Indeed, they gave a counter-example of a convex problem satisfying both Assumptions 4.1 and 4.2, but for which the subproblems (4.16) and (4.17) are not solvable. This implies that the convergence analysis in [Boyd et al., 2011] is erroneous. According to [Chen et al., 2017], the following additional assumption is needed.

Assumption 4.3. *The subproblems of ADMM,* i.e. (4.16) *and* (4.17)*, are solvable and have non-empty bounded solution sets.*

We are now ready to state the main convergence results of ADMM.

Theorem 4.1. *Under Assumptions 4.1, 4.2 and 4.3, the iterates of ADMM (Algorithm 4.1) for solving problem* (4.1) *satisfy the following:*

(a) $f(\mathbf{x}^{(k)}) + g(\mathbf{z}^{(k)})$ *converges to the optimal value of* (4.1)*.*

(b) $\mathbf{y}^{(k)}$ *converges to an optimal solution to the dual problem* (4.4)*.*

(c) $\mathbf{r}^{(k)} := \mathbf{A}\mathbf{x}^{(k)} + \mathbf{B}\mathbf{z}^{(k)} - \mathbf{c}$*, called the* residual*, converges to* $\mathbf{0}$*.*

(d) Any limit point[4] $(\mathbf{x}^*, \mathbf{z}^*, \mathbf{y}^*)$ *of the sequence* $(\mathbf{x}^{(k)}, \mathbf{z}^{(k)}, \mathbf{y}^{(k)})$ *is a solution to the the KKT system* (4.22)–(4.24)*,* i.e. $(\mathbf{x}^*, \mathbf{z}^*, \mathbf{y}^*)$ *is a primal-dual optimal solution to* (4.1)*. In addition,* $\mathbf{A}\mathbf{x}^{(k)}$ *converges to* $\mathbf{A}\mathbf{x}^*$ *and* $\mathbf{B}\mathbf{z}^{(k)}$ *converges to* $\mathbf{B}\mathbf{z}^*$*.*

(e) The rate of convergence of all the above points is at least $\mathcal{O}(1/\sqrt{k})$*.*

A proof of (a), (b), (c) can be found in *e.g.* [Boyd et al., 2011] or [Chen et al., 2017], and a proof of (d) can be found in [Chen et al., 2017]. It should be noted that the above convergence results also hold for a generalized variant of ADMM, where the $\mathbf{y}$-update step (4.18) is replaced by

$$\mathbf{y}^{(k+1)} \leftarrow \mathbf{y}^{(k)} + \alpha\rho(\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{z}^{(k+1)} - \mathbf{c}), \tag{4.25}$$

where $\alpha \in \left(0, \frac{1+\sqrt{5}}{2}\right)$ is called the *step-size*.

Regarding the rate of convergence (e), a proof and addition results can be found in *e.g.* [Wang and Banerjee, 2013, He and Yuan, 2015, Davis and Yin, 2016]. Without further assumptions, [Davis and Yin, 2016] proved that the $\mathcal{O}(1/\sqrt{k})$ rate of convergence is actually tight. One can also obtain an $\mathcal{O}(1/k)$ rate in the ergodic sense (*i.e.* taking a convex combination of all the iterates). If further assumptions hold, *e.g.* strong convexity or Lipschitz differentiability, then better rates of convergence (*e.g.* linear) can be obtained. We refer to *e.g.* [Deng and Yin, 2016, Hong and Luo, 2017, Giselsson, 2017] and references therein for more details.

[4] A vector $\mathbf{u}$ is a limit point of a sequence $(\mathbf{u}^{(k)})$ if there exists a subsequence of $(\mathbf{u}^{(k)})$ that converges to $\mathbf{u}$. A limit point is also called an *accumulation point* ou a *cluster point*. Note that the sequence may be divergent and may have multiple limit points. When it is convergent, $\mathbf{u}$ becomes its *limit* (and of course its only limit point).

4.2 BEYOND TWO-BLOCK, SEPARABLE AND CONVEX PROBLEMS

4.2.1 *Multi-block problems*

A natural and straightforward extension of ADMM can be applied to solving problems with more than two blocks of variables:

$$\begin{aligned} \min \quad & f_1(\mathbf{x}_1) + f_2(\mathbf{x}_2) + \cdots + f_D(\mathbf{x}_D) \\ \text{s.t.} \quad & \mathbf{A}_1\mathbf{x}_1 + \mathbf{A}_2\mathbf{x}_2 + \cdots + \mathbf{A}_D\mathbf{x}_D = \mathbf{c}, \end{aligned} \tag{4.26}$$

where $\mathbf{c} \in \mathbb{R}^p$ and $\forall d = 1, 2, \ldots, D : \mathbf{x}_d \in \mathbb{R}^{n_d}, \mathbf{A}_d \in \mathbb{R}^{p \times n_d}$ and $f_d : \mathbb{R}^{n_d} \to \mathbb{R} \cup \{+\infty\}$ are extended real-valued functions. The augmented Lagrangian for this problem is

$$L_\rho(\mathbf{x}_1, \ldots, \mathbf{x}_D, \mathbf{y}) = \sum_{d=1}^{D} f_d(\mathbf{x}_d) + \mathbf{y}^\top \left(\sum_{d=1}^{D} \mathbf{A}_d\mathbf{x}_d - \mathbf{c} \right) + \frac{\rho}{2} \left\| \sum_{d=1}^{D} \mathbf{A}_d\mathbf{x}_d - \mathbf{c} \right\|_2^2, \tag{4.27}$$

and the corresponding ADMM algorithm is presented in Algorithm 4.2.

ALGORITHM 4.2 ADMM for solving multi-block problems.

1: Initialization: $k \leftarrow 0$, $\mathbf{x}_d^{(0)} \in \mathbb{R}^{n_d}$ for $d = 1, \ldots, D$, and $\mathbf{y}^{(0)} \in \mathbb{R}^p$.

2: For $d = 1, 2, \ldots, D$, update $\mathbf{x}_d^{(k+1)}$ as

$$\mathbf{x}_d^{(k+1)} \leftarrow \operatorname*{argmin}_{\mathbf{x}_d} L_\rho(\mathbf{x}_{[1,d-1]}^{(k+1)}, \mathbf{x}_d, \mathbf{x}_{[d+1,D]}^{(k)}, \mathbf{y}^{(k)}). \tag{4.28}$$

3: Update $\mathbf{y}^{(k+1)}$ as

$$\mathbf{y}^{(k+1)} \leftarrow \mathbf{y}^{(k)} + \rho \left(\sum_{d=1}^{D} \mathbf{A}_d\mathbf{x}_d^{(k+1)} - \mathbf{c} \right). \tag{4.29}$$

Let $k \leftarrow k + 1$ and go to Step 2.

Unlike for two-block problems, the *convexity* of the objective functions $(f_d)_{d=1}^D$ is not sufficient to ensure convergence of ADMM. Indeed, [Chen et al., 2016] gave a concrete example of a three-block problem for which ADMM is guaranteed to diverge for any penalty $\rho > 0$ and any starting point in a continuously dense half-space of dimension 3.

With further assumptions, convergence of ADMM for multi-block problems can be guaranteed. For example, for three-block problems, [Chen et al., 2016] showed that ADMM is convergent for any $\rho > 0$ if two of the coefficient matrices $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3$ are orthogonal, *i.e.* either $\mathbf{A}_1^\top\mathbf{A}_2 = \mathbf{0}$ or $\mathbf{A}_2^\top\mathbf{A}_3 = \mathbf{0}$ or $\mathbf{A}_1^\top\mathbf{A}_3 = \mathbf{0}$. Moreover, in this case, ADMM achieves a worst-case $\mathcal{O}(1/k)$ rate of convergence in the ergodic sense.

In the case of $(f_d)_{d=1}^D$ being **strongly convex** with parameters $(\sigma_d)_{d=1}^D$ (respectively), [Han and Yuan, 2012] showed that ADMM is convergent if the penalty parameter ρ satisfies

$$0 < \rho < \min_{1 \le d \le D} \frac{2\sigma_d}{3(D-1)\|\mathbf{A}_d\|^2}. \tag{4.30}$$

Similar results were proposed by [Lin et al., 2015] for the case where only $(D-1)$ functions among $(f_d)_{d=1}^D$ are strongly convex. They also provided an ergodic rate of convergence of $\mathcal{O}(1/k)$, and a non-ergodic rate of convergence of $o(1/k)$. Note that a condition on ρ (such as the above) is necessary, as shown by [Chen et al., 2016]. In fact, they gave an example of strongly convex functions for which ADMM is divergent for a certain value of ρ.

4.2.2 Nonseparable problems

So far we have mentioned only problems with a separable objective, *i.e.* a sum of independent terms over each block of variables. A direct extension of problem (4.26) to nonseparable case is the following:

$$\begin{aligned} \min \quad & h(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_D) + f_1(\mathbf{x}_1) + f_2(\mathbf{x}_2) + \cdots + f_D(\mathbf{x}_D) \\ \text{s.t.} \quad & \mathbf{A}_1\mathbf{x}_1 + \mathbf{A}_2\mathbf{x}_2 + \cdots + \mathbf{A}_D\mathbf{x}_D = \mathbf{c}, \end{aligned} \tag{4.31}$$

where the notations and conditions are the same as in (4.26), except that we have added a coupling term $h(\mathbf{x}_1, \dots, \mathbf{x}_D)$ to the objective. Here $h(\mathbf{x}_1, \dots, \mathbf{x}_D)$ is supposed to be **convex** jointly with respect to $(\mathbf{x}_d)_{d=1}^D$. We refer to Section 4.2.3 for a discussion on the general nonseparable and nonconvex case.

The standard ADMM algorithm for solving (4.31) is the same as Algorithm 4.2, except that the Lagrangian $L_\rho(\mathbf{x}_1, \dots, \mathbf{x}_D, \mathbf{y})$ now contains $h(\mathbf{x}_1, \dots, \mathbf{x}_D)$ in addition to (4.27). The introduction of this coupling term makes it really difficult to analyze the convergence of ADMM, even for two-block problems, *i.e.* $D = 2$. As a result, many authors tend to study different extensions of ADMM instead. For example, [Hong et al., 2014] and [Cui et al., 2015] studied the convergence of an extension of ADMM, called *majorization-minimization ADMM*, where at each $\mathbf{x}$-update step (4.28), the nonseparable term $h(\mathbf{x}_1, \dots, \mathbf{x}_D)$ is replaced by its upper-bound approximations. Another extension of ADMM, called *proximal ADMM*, was also studied by *e.g.* [Gao and Zhang, 2017] (for two-block problems) and [Chen et al., 2015] (for h being a quadratic function). In proximal ADMM, a proximal term is added to the right-hand side of the $\mathbf{x}$-update step (4.28) of the standard ADMM. We will discuss these kinds of extension for ADMM in more detail in Section 4.3.

4.2.3 Nonconvex problems

While ADMM was originally designed for solving convex problems, it has recently been applied to solve a wide range of nonconvex problems with excellent performance (see *e.g.* [Hong et al., 2016] and references therein). Therefore, studying the behavior of nonconvex ADMM has become an important research topic. Due to the hardness of the nonconvexity, the amount of published work on analyzing the convergence of the nonconvex standard ADMM is quite limited compared to the convex case. We present such analyses in the subsections below.

Separable two-block case

The problem of interest here is (4.1) without the convexity assumption. The standard ADMM for this problem is the same as Algorithm 4.1. We should note that the

subproblems (4.16) and (4.17) are possibly no longer convex. Therefore, for ADMM to be meaningful in this case, we implicitly assume that its subproblems can be solved to global optimality (idem for the multi-block case in Section 4.2.3).

For the special case of the problem where the matrix $\mathbf{B}$ is identity, [Li and Pong, 2015] proposed an extension of ADMM where a proximal term based on a Bregman divergence[5] is added to the $\mathbf{z}$-update step. This proximal term can be discarded to obtain the standard ADMM. They proved that the iterates of this algorithm converge to a primal-dual stationary point, under the assumptions that the functions f and g are semi-algebraic, g is twice continuously differentiable (hence ∇g is Lipschitz continuous) with uniformly bounded Hessian, $\mathbf{A}^\top \mathbf{A} \succeq \mu \mathbf{I}$ for some $\mu > 0$, and the penalty parameter ρ is larger than a certain value. Later, [Guo et al., 2017] proved similar results but with weaker assumptions: g only needs to be Lipschitz differentiable and no constraint on its Hessian is needed.

For the general case, [Gonçalves et al., 2017] studied the convergence of another proximal ADMM (in which the proximal terms can be discarded, when $\mathbf{B}$ is invertible, to obtain the standard ADMM). They proved that this algorithm is convergent under even weaker assumptions compared to [Guo et al., 2017]. However, when reducing to standard ADMM for $\mathbf{B}$ being identity, their range for ρ is worse than that of [Guo et al., 2017]. Very recently, [Themelis and Patrinos, 2017] presented an analysis for a nonconvex ADMM with *over-relaxation*, which includes the standard ADMM as a special case (*c.f.* Section 4.3.2 for details on over-relaxed ADMM). Their results are perhaps the strongest compared to the previous work: they proved the convergence of ADMM under weaker assumptions, yet with a better range of the penalty parameter ρ. In Figure 4.1, we compare the ranges of ρ obtained by all the above analyses for the standard ADMM applied to the special case of the problem where $\mathbf{A}$ is full rank and $\mathbf{B}$ is identity.

Multi-block case

This general case concerns the problems (4.26) (for multiple blocks), or (4.31) (for multiple and nonseparable blocks), except that the convexity assumption is removed. Obviously the standard ADMM for these problems is the same as Algorithm 4.2, for which we assume that the subproblems (4.28) can be solved to global optimality.

Since analyzing the standard ADMM is hard, some authors chose to deal with different variants. For example, [Wang et al., 2015a] studied the convergence of a proximal ADMM based on a Bregman divergence for three-block separable objectives. For the general problem (4.31) (without convexity), [Jiang et al., 2016] studied another proximal ADMM based on norms with respect to matrices. They show that the algorithm converges to an approximate ϵ-stationary solution for a certain range of penalty parameters, under the assumptions that h and f_D are Lipschitz differentiable.

[Hong et al., 2016] studied the standard ADMM for the consensus and sharing problems, which are special cases of (4.31). The proved that the algorithm converges to a stationary point for a large enough penalty parameter, under the assumptions that $D-1$ functions in the objective are convex (but possibly nonsmooth), and the other is nonconvex but Lipschitz differentiable.

[5] We discuss in more detail this kind of extension in Section 4.3.

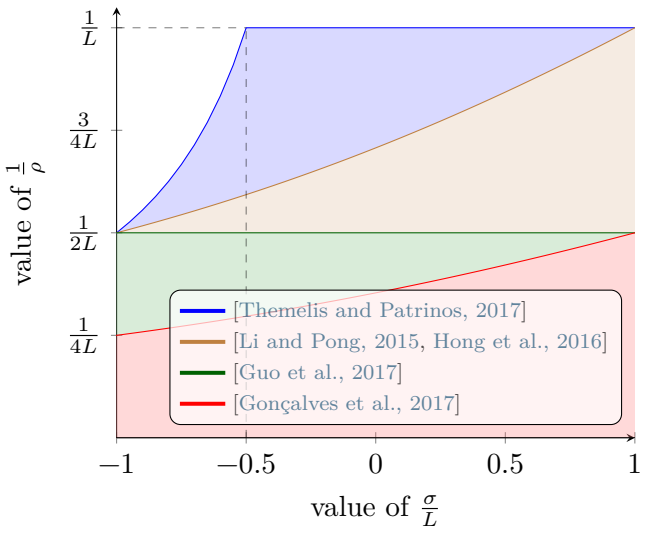


FIGURE 4.1 Comparison of recent convergence analyses of ADMM for nonconvex objectives. This plot shows the supremum of the inverse of the penalty parameter, *i.e.* $\frac{1}{\rho}$, for which the standard ADMM is convergent for solving the special case of (4.1) without convexity where $\mathbf{A}$ is full rank and $\mathbf{B}$ is identity, under the assumption that g is Lipschitz differentiable. Let L denote the corresponding Lipschitz constant, then there exists $\sigma \in [-L, L]$ such that $\sigma\|\mathbf{z}-\mathbf{t}\|^2 \leq \langle \nabla g(\mathbf{z}) - \nabla g(\mathbf{t}), \mathbf{z}-\mathbf{t}\rangle \leq L\|\mathbf{z}-\mathbf{t}\|^2 \ \forall \mathbf{z}, \mathbf{t} \in \mathbb{R}^m$. The parameter σ is called the convexity constant of g (if $\sigma = 0$ then g is convex, if $\sigma > 0$ then g is strongly convex, and if $\sigma < 0$ then g is not convex but $g(\cdot) - \frac{\sigma}{2}\|\cdot\|^2$ is). Intuitively, the higher σ is, the easier the problem is and thus we should expect a larger range of ρ for which ADMM is convergent. This is the case for most of the analyses, as shown in the figure. Note that the range provided by [Themelis and Patrinos, 2017] is tight, *i.e.* no better range can be obtained. Figure reproduced from [Themelis and Patrinos, 2017] with permission.

Perhaps the current most prominent convergence analysis of standard ADMM for nonseparable multi-block problems is due to [Wang et al., 2015b]. Under a number of assumptions (including Lipschitz differentiability), they showed that ADMM converges to a stationary point of (4.31). We give a brief summary of their results in Table 4.1.

4.3 OTHER EXTENSIONS AND VARIATIONS

Many extensions and variations of ADMM have been explored in the literature. In this section, we briefly survey some of these variants. A similar treatment can be found in [Boyd et al., 2011], though the references therein are more dated.

4.3.1 *Adaptive penalty parameter*

A standard extensions is to use different penalty parameter $\rho^{(k)}$ at each iteration, which can accelerate the convergence and make the algorithm less dependent on the initial parameter.

The most popular adaptive scheme is *residual balancing*, due to [He et al., 2000]

TABLE 4.1 A summary of sufficient conditions for ADMM to be convergent for solving (4.31) without convexity, according to the analysis of [Wang et al., 2015b]. The penalty parameter ρ is implicitly chosen to be large enough. There are two scenarios considered: first, if only part of the objective function is Lipschitz differentiable, then the nonseparable term and the last separable term have to be so, and further assumptions on the other terms are required (Scenario 1 below); second, if the objective is Lipschitz differentiable then no further assumptions on it is required (Scenario 2 below). It should be noted that in Scenario 1, the nonseparable term does not contain the last block, and the functions $f_0, \dots, f_{D-1}$ are not required to exist. We refer to [Wang et al., 2015b] for further details.

<table>
<tr><td></td><td colspan="2">Scenario 1</td><td>Scenario 2</td></tr>
<tr><td>objective</td><td colspan="2">$h(\mathbf{x}_1, \dots, \mathbf{x}_{D-1}) + \sum_{d=1}^{D} f_D(\mathbf{x}_D)$
coercive over $\sum_{d=1}^{D} \mathbf{A}_d \mathbf{x}_d = \mathbf{0}$</td><td rowspan="5">The objective is Lipschitz differentiable</td></tr>
<tr><td>h, f_D</td><td colspan="2">Lipschitz differentiable</td></tr>
<tr><td></td><td>Scenario 1a</td><td>Scenario 1b</td></tr>
<tr><td>f_1</td><td>lower semi-continuous</td><td>∂f_1 bounded in any bounded set</td></tr>
<tr><td>$(f_d)_{d=2}^{D-1}$</td><td>restricted prox-regular</td><td>piecewise linear</td></tr>
<tr><td>$(\mathbf{A}_d)_{d=1}^{D}$</td><td colspan="3">Feasibility: $\mathrm{Im}([\mathbf{A}_1 \quad \dots \quad \mathbf{A}_{D-1}]) \subseteq \mathrm{Im}(\mathbf{A}_D)$</td></tr>
<tr><td></td><td colspan="3">solution to each ADMM subproblem is unique and is Lipschitz w.r.t. the input</td></tr>
</table>

and [Wang and Liao, 2001]:

$$\rho^{(k+1)} = \begin{cases} \alpha\rho^{(k)} & \text{if } \|\mathbf{r}^{(k)}\|_2 > \mu\|\mathbf{s}^{(k)}\|_2, \\ \rho^{(k)}/\beta & \text{if } \|\mathbf{s}^{(k)}\|_2 > \mu\|\mathbf{r}^{(k)}\|_2, \\ \rho^{(k)} & \text{otherwise,} \end{cases} \tag{4.32}$$

where $\mu > 1, \alpha > 1$ and $\beta > 1$ are parameters. The idea behind this is to keep a balance between the norms of the primal and dual residuals (so that they are always within a factor μ of one another). Since the algorithm converges only if both of them converge to zero, keeping them decrease together can accelerate the convergence.

More recently, [Xu et al., 2017] proposed a spectral adaptive scheme, motivated by the dual formulation of ADMM (called the Douglas-Rachdord splitting algorithm). They showed that their scheme outperforms residual balancing on a variety of problems.

4.3.2 *Over-relaxation*

In the $\mathbf{z}$-update (4.17) and the $\mathbf{y}$-update (4.18), the quantity $\mathbf{A}\mathbf{x}^{(k+1)}$ can be replaced with $\alpha^{(k)}\mathbf{A}\mathbf{x}^{(k+1)} - (1-\alpha^{(k)})(\mathbf{B}\mathbf{z}^k - \mathbf{c})$, *i.e.* these steps become:

$$\mathbf{z}^{(k+1)} \leftarrow \underset{\mathbf{z}}{\operatorname{argmin}} \left\{ g(\mathbf{z}) + \langle \mathbf{y}^{(k)}, \mathbf{B}\mathbf{z} - \mathbf{c} \rangle + \frac{\rho}{2} \|\mathbf{t}^{(k+1)} + \mathbf{B}\mathbf{z} - \mathbf{c}\|_2^2 \right\}, \tag{4.33}$$

$$\mathbf{y}^{(k+1)} \leftarrow \mathbf{y}^{(k)} + \rho(\mathbf{t}^{(k+1)} + \mathbf{B}\mathbf{z}^{(k+1)} - \mathbf{c}), \tag{4.34}$$

where

$$\mathbf{t}^{(k+1)} = \alpha^{(k)}\mathbf{A}\mathbf{x}^{(k+1)} - (1-\alpha^{(k)})(\mathbf{B}\mathbf{z}^k - \mathbf{c}). \tag{4.35}$$

The parameter $\alpha^{(k)} > 0$ is called *relaxation parameter*.

Over-relaxed ADMM with $\alpha^{(k)} > 1$ has been shown to converge faster than non-relaxed ADMM on different practical problems (see *e.g.* [Eckstein and Bertsekas, 1992] and [Eckstein, 1994]). Its theoretical convergence for convex problems was analyzed by *e.g.* [Nishihara et al., 2015], and by *e.g.* [Themelis and Patrinos, 2017] for nonconvex ones.

4.3.3 More general augmenting terms

The ℓ_2 norm in the augmented Lagrangian (4.10) (or (4.27) for multiple blocks) can be replaced by a more general distance function, for example:

$$L_{\mathbf{M}}(\mathbf{x}, \mathbf{z}, \mathbf{y}) = f(\mathbf{x}) + g(\mathbf{z}) + \mathbf{y}^\top(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}) + \frac{1}{2}\|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}\|_{\mathbf{M}}^2, \tag{4.36}$$

where $\|\mathbf{u}\|_{\mathbf{M}}$ denotes the Mahalanobis norm, defined by $\|\mathbf{u}\|_{\mathbf{M}} = \sqrt{\langle \mathbf{u}, \mathbf{M}\mathbf{u}\rangle}$ where $\mathbf{M}$ is a positive definite matrix. When $\mathbf{M} = \rho\mathbf{I}$ then the above becomes the usual ℓ_2 augmented Lagrangian. Even more generally, at each iteration of ADMM, the matrix $\mathbf{M}$ can be allowed to vary, *i.e.* taking a value $\mathbf{M}^{(k)}$. The convergence of such algorithm was analyzed in *e.g.* [He et al., 2002].

The Mahalanobis norm is a special case of an even more general distance function called the *Bregman divergence* [Bregman, 1967, Censor and Zenios, 1997]. The Bregman divergence induced by a continuously differentiable and strictly convex function ϕ is defined by

$$D_\phi(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) - \phi(\mathbf{y}) - \langle \nabla\phi(\mathbf{y}), \mathbf{x} - \mathbf{y}\rangle\,. \tag{4.37}$$

Since ϕ is convex we have $D_\phi(\mathbf{x}, \mathbf{y}) \geq 0\ \forall \mathbf{x}, \mathbf{y}$ and equality occurs if and only if $\mathbf{x} = \mathbf{y}$. As an example, the ℓ_2-norm is the Bregman divergence induced by the function $\phi(\mathbf{x}) := \|\mathbf{x}\|_2^2$. Bregman ADMM was analyzed in *e.g.* [Wang and Banerjee, 2014].

4.3.4 Proximal ADMM

This is another popular variant of ADMM, in which a proximal term is added to each subproblem:

$$\mathbf{x}_d^{(k+1)} = \operatorname*{argmin}_{\mathbf{x}_d} \left\{ L_\rho(\mathbf{x}_{[1,d-1]}^{(k+1)}, \mathbf{x}_d, \mathbf{x}_{[d+1,D]}^{(k)}, \mathbf{y}^{(k)}) + \Delta_d(\mathbf{x}, \mathbf{x}_d^{(k)}) \right\}, \tag{4.38}$$

where $\Delta_d(\cdot, \cdot)$ is some distance function. This distance function can be *e.g.* the Euclidean distance (*i.e.* ℓ_2 norm), matrix based norm, or Bregman divergence. In addition, the augmented Lagrangian in the above update can also be replaced by a more general augmented term, as discussed previously in Section 4.3.3.

The convergence of proximal ADMM was studied by *e.g.* [Wang and Banerjee, 2014, Deng and Yin, 2016] for convex problems, and more recently by *e.g.* [Li and Pong, 2015, Wang et al., 2015a, Guo et al., 2017, Gonçalves et al., 2017] for nonconvex problems.

5

Alternating Direction Graph Matching

This chapter presents our first major contribution. We introduce a graph matching method that can account for constraints of arbitrary order, with arbitrary potential functions. Unlike previous decomposition approaches that rely on the graph structures, we introduce a decomposition of the matching constraints. Graph matching is then reformulated as a non-convex non-separable optimization problem that can be split into smaller and much-easier-to-solve subproblems, by means of ADMM. The proposed framework is modular, scalable, and can be instantiated into different variants. Two instantiations are studied exploring pairwise and higher-order constraints. Experimental results on widely adopted benchmarks involving synthetic and real examples demonstrate that the proposed solutions outperform existing pairwise graph matching methods, and competitive with the state of the art in higher-order settings. A preliminary version of this work was published in [Lê-Huu and Paragios, 2017].

5.1 CONTEXT AND MOTIVATION

The proposed method is motivated by two main factors: 1) the recent rise of decomposition methods in computer vision, and 2) the limitations of current state-of-the-art graph matching methods.

Decomposition is a general approach to solving a problem by breaking it up into smaller ones that can be efficiently addressed separately, and then reassembling the results towards a globally consistent solution of the original non-decomposed problem [Bertsekas, 1999]. In computer vision, decomposition methods such as dual decomposition and ADMM have been applied to optimizing MRFs [Komodakis et al., 2011, Martins et al., 2015] and to solving graph/hypergraph matching [Torresani et al., 2013, Zeng et al., 2010]. The main idea is to decompose the original complex graph into simpler subgraphs and then reassembling the solutions on these subgraphs using different mechanisms. While in MRF inference, this concept has been proven to be flexible and powerful, that is far from being the case in graph matching, due to the hardness of the one-to-one constraints (*c.f.* Chapter 3). Indeed, to deal with these constraints, [Torresani et al., 2013] for example adopted a strategy that creates subproblems that are also smaller graph matching problems, which are computationally highly challenging. Moreover, subgradient method has been used to impose consensus, which is known to have slow rate of convergence [Bertsekas, 1999]. Therefore, dual decomposition is a very slow method and works for a limited set of energy models often associated with small sizes and low to medium geometric connectivities [Torresani et al., 2013].

On the other hand, different methods for graph matching have been recently proposed with excellent performance. The current state-of-the-art method is a block coordinate ascent algorithm proposed by [Nguyen et al., 2015]. Despite the impressive performance, this method has two limitations: (a) it cannot be applied to graph matching of arbitrary order other than third and fourth, and (b) it cannot deal with graph matching where occlusion is allowed on both sides, nor with one-to-many or many-to-many matching.

In this work, we propose a novel class of decomposition algorithms that can overcome all the aforementioned limitations. These methods work with arbitrary potentials of any order and with any matching constraints (one-to-one, one-to-many, or else). Yet, they are also very computationally efficient.

5.2 GENERAL DECOMPOSITION FRAMEWORK FOR GRAPH MATCHING

In this section, we introduce a general decomposition framework for graph matching, by means of ADMM.

First, let us recall some notations and formulations from Chapter 3 (Section 3.2.2). The solution of graph matching is represented by an assignment matrix $\mathbf{X} \in \{0,1\}^{n_1 \times n_2}$, where n_1, n_2 are the numbers of nodes of the graphs. Depending on application, $\mathbf{X}$ may obey different matching constraints. For example, the following set represents the common *one-to-(at most)-one* constraints:

$$\overline{\mathcal{M}}_{\text{one-to-one}} = \left\{ \mathbf{X} \in \{0,1\}^{n_1 \times n_2} \;\middle|\; \begin{array}{l} \text{sum of each row of } \mathbf{X} \text{ is } \leq 1 \\ \text{sum of each column of } \mathbf{X} \text{ is } \leq 1 \end{array} \right\}. \tag{5.1}$$

If no occlusion is allowed then "≤ 1" is replaced by "$= 1$". Let $\overline{\mathcal{M}}$ denote general matching constraints whose type will be specified or understood from the context. We will be working mostly with the row-wise vectorized replica of $\mathbf{X}$, which is the assignment vector $\mathbf{x} = \text{vec}(\mathbf{X}) \in \mathbb{R}^n$, where $n = n_1 n_2$. Instead of writing $\text{mat}(\mathbf{x}) \in \overline{\mathcal{M}}$ to represent the matching constraints, we write $\mathbf{x} \in \overline{\mathcal{X}}$ where

$$\overline{\mathcal{X}} = \left\{ \mathbf{x} \in \mathbb{R}^{n_1 \times n_2} \;\middle|\; \text{mat}(\mathbf{x}) \in \overline{\mathcal{M}} \right\}. \tag{5.2}$$

General graph matching can then be formulated as follows.

Problem 5.1 (D^{th}-order graph matching)**.** *Minimize*

$$F^1(\mathbf{x}) + F^2(\mathbf{x}, \mathbf{x}) + \cdots + F^D(\mathbf{x}, \mathbf{x}, \dots, \mathbf{x}) \tag{5.3}$$

subject to $\mathbf{x} \in \overline{\mathcal{X}}$*, where* F^d $(d = 1, \dots, D)$ *is the multilinear form of a tensor* $\mathbf{F}^d$ *representing the* d^{th}*-order potentials, defined by:*

$$F^d(\mathbf{x}_1, \dots, \mathbf{x}_d) = \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} F_{i_1 i_2 \dots i_d} x^1_{i_1} x^2_{i_2} \cdots x^d_{i_d}, \tag{5.4}$$

where $\mathbf{x}_j = (x^j_1, x^j_2, \dots, x^j_{n_j}) \in \mathbb{R}^{n_j}$ *for* $j = 1, 2, \dots, d$.

Next, we propose a decomposition framework for solving the *continuous relaxation*

of Problem 5.1, *i.e.* minimizing (5.3) subject to $\mathbf{x} \in \mathcal{X}$, where $\mathcal{X}$ is the same as $\overline{\mathcal{X}}$ except that the binary constraint is replaced by $\mathbf{0} \leq \mathbf{x} \leq \mathbf{1}$. For example, the relaxed one-to-one constraints are represented by

$$\mathcal{X}_{\text{one-to-one}} = \left\{ \mathbf{x} \in [0,1]^{n_1 \times n_2} \;\middle|\; \begin{array}{l} \text{sum of each row of mat}(\mathbf{x}) \text{ is} \leq 1 \\ \text{sum of each column of mat}(x) \text{ is} \leq 1 \end{array} \right\}. \tag{5.5}$$

Once a continuous solution has been found, it can be converted to a discrete one using *e.g.* the Hungarian method [Kuhn, 1955].

As discussed previously in Section 5.1, the one-to-one constraint makes the problem hard to solve. To deal with these constraints, [Torresani et al., 2013] adopted a strategy that creates subproblems that are also smaller graph matching problems, which are computationally highly challenging. In our proposed framework, we do not rely on the structure of the graphs but instead, on the nature of the variables. In fact, the idea is to decompose the assignment vector $\mathbf{x}$ (by means of Lagrangian relaxation) into different variables where each variable obeys weaker constraints (that are easier to handle). For example, instead of dealing with the assignment vector $\mathbf{x} \in \mathcal{X}$, we can represent it by two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$, where the sum of each row of $\text{mat}(\mathbf{x}_1)$ is ≤ 1 and the sum of each column of $\text{mat}(\mathbf{x}_2)$ is ≤ 1, and we constrain these two vectors to be equal. More generally, we can decompose $\mathbf{x}$ into as many vectors as we want, and in any manner, the only condition is that the set of constraints imposed on these vectors must be equivalent to $\mathbf{x}_1 = \mathbf{x}_2 = \cdots = \mathbf{x}_p \in \mathcal{X}$ where p is the number of vectors. As for the objective function (5.3), there is also an infinite number of ways to re-write it under the new variables $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p$. The only condition is that the re-written objective function must be equal to the original one when $\mathbf{x}_1 = \mathbf{x}_2 = \cdots = \mathbf{x}_p = \mathbf{x}$. For example, if $p = D$ then one can re-write (5.3) as

$$F^1(\mathbf{x}_1) + F^2(\mathbf{x}_1, \mathbf{x}_2) + \cdots + F^D(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_D). \tag{5.6}$$

Each combination of (a) such a variable decomposition and (b) such a way of re-writing the objective function will yield a different Lagrangian relaxation and thus, produce a different algorithm. Since there are virtually infinite of such combinations, the number of algorithms one can design from them is also unlimited, not to mention the different choices of the reassembly mechanism, such as subgradient methods [Shor et al., 1985], cutting plane methods [Bertsekas, 1999], ADMM (Chapter 4), or others. We call the class of algorithms that are based on ADMM **Alternating Direction Graph Matching (ADGM)** algorithms. A major advantage of ADMM over the other mechanisms is that its subproblems involve only one block of variables, regardless of the form the objective function.

As an illustration of ADGM, we present below a particular example. Nevertheless, this example is still general enough to include an infinite number of special cases. Indeed, it is straightforward to see that the continuous relaxation of Problem 5.1 is equivalent to the following problem.

Problem 5.2 (Decomposed graph matching)**.** *Minimize*

$$F^1(\mathbf{x}_1) + F^2(\mathbf{x}_1, \mathbf{x}_2) + \cdots + F^D(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_D) \tag{5.7}$$

subject to

$$\mathbf{A}_1\mathbf{x}_1 + \mathbf{A}_2\mathbf{x}_2 + \cdots + \mathbf{A}_D\mathbf{x}_D = \mathbf{0}, \tag{5.8}$$
$$\mathbf{x}_d \in \mathcal{X}_d \quad \forall\ 1 \leq d \leq D, \tag{5.9}$$

where $(\mathbf{A}_d)_{1\leq d\leq D}$ *are* $m \times n$ *matrices, defined in such a way that* (5.8) *is equivalent to* $\mathbf{x}_1 = \mathbf{x}_2 = \cdots = \mathbf{x}_D$*, and* $(\mathcal{X}_d)_{1\leq d\leq D}$ *are closed convex subsets of* $\mathbb{R}^n$ *satisfying*

$$\mathcal{X}_1 \cap \mathcal{X}_2 \cap \cdots \cap \mathcal{X}_D = \mathcal{X}. \tag{5.10}$$

We have decomposed $\mathbf{x}$ into D vectors $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_D)$. This formulation allows direct application of ADMM.

The augmented Lagrangian of Problem 5.2 is given by

$$L_\rho(\mathbf{x}_1, \ldots, \mathbf{x}_D, \mathbf{y}) = \sum_{d=1}^{D} F^d(\mathbf{x}_1, \ldots, \mathbf{x}_d) + \mathbf{y}^\top \left(\sum_{d=1}^{D} \mathbf{A}_d\mathbf{x}_d\right) + \frac{\rho}{2}\left\|\sum_{d=1}^{D} \mathbf{A}_d\mathbf{x}_d\right\|_2^2. \tag{5.11}$$

Recall from Chapter 4 that standard ADMM solves Problem 5.2 by iterating:

1. For $d = 1, 2, \ldots, D$, update $\mathbf{x}_d$:

$$\mathbf{x}_d^{(k+1)} = \operatorname*{argmin}_{\mathbf{x}\in\mathcal{X}_d} L_\rho(\mathbf{x}_1^{(k+1)}, \ldots, \mathbf{x}_{d-1}^{(k+1)}, \mathbf{x}, \mathbf{x}_{d+1}^{(k)}, \ldots, \mathbf{x}_D^{(k)}, \mathbf{y}^{(k)}). \tag{5.12}$$

2. Update $\mathbf{y}$:

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + \rho\left(\sum_{d=1}^{D} \mathbf{A}_d\mathbf{x}_d^{(k+1)}\right). \tag{5.13}$$

The algorithm converges if the following *residual* converges to 0 as $k \to \infty$:

$$r^{(k)} = \left\|\sum_{d=1}^{D} \mathbf{A}_d\mathbf{x}_d^{(k)}\right\|_2^2 + \sum_{d=1}^{D} \left\|\mathbf{A}_d\mathbf{x}_d^{(k)} - \mathbf{A}_d\mathbf{x}_d^{(k-1)}\right\|_2^2. \tag{5.14}$$

We will discuss the convergence of ADMM for graph matching later in Section 5.3.5.

The $\mathbf{y}$-update step (5.13) and the computation of the residual (5.14) is trivial. Let us focus on the $\mathbf{x}$-update step (5.12), *i.e.* the so-called subproblems. In this step, all variable blocks are fixed except one. It can be observed that the objective function (5.7) is linear with respect to each variable block. Therefore, (5.12) involves a sum of a linear function and a penalty term.

Indeed, from equation (3.11) (page 17) we have

$$F^i(\mathbf{x}_{[1,d-1]}^{(k+1)}, \mathbf{x}, \mathbf{x}_{[d+1,i]}^{(k)}) = \mathcal{F}^i \bigotimes_{j=1}^{d-1} \mathbf{x}_j^{(k+1)} \otimes_d \mathbf{x} \bigotimes_{l=d+1}^{i} \mathbf{x}_l^{(k)}. \tag{5.15}$$

As a reminder, the $\bigotimes$ notation is defined in equation (3.10) (page 17). We have also used the notation $\mathbf{x}_{[a,b]}$ to denote $(\mathbf{x}_a, \mathbf{x}_{a+1}, \ldots, \mathbf{x}_b)$ (by convention, if $a > b$ then $\mathbf{x}_{[a,b]}$

is ignored). Regrouping the above equation we get:

$$\sum_{i=d}^{D} F^i(\mathbf{x}_{[1,d-1]}^{(k+1)}, \mathbf{x}, \mathbf{x}_{[d+1,i]}^{(k)}) = \sum_{i=d}^{D} \left(\mathcal{F}^i \bigotimes_{j=1}^{d-1} \mathbf{x}_j^{(k+1)} \otimes_d \mathbf{x} \bigotimes_{l=d+1}^{i} \mathbf{x}_l^{(k)} \right) \tag{5.16}$$

$$= \left(\sum_{i=d}^{D} \mathcal{F}^i \bigotimes_{j=1}^{d-1} \mathbf{x}_j^{(k+1)} \bigotimes_{l=d+1}^{i} \mathbf{x}_l^{(k)} \right) \cdot \mathbf{x}, \tag{5.17}$$

which is clearly a linear function with respect to $\mathbf{x}$:

$$\sum_{i=d}^{D} F^i(\mathbf{x}_{[1,d-1]}^{(k+1)}, \mathbf{x}, \mathbf{x}_{[d+1,i]}^{(k)}) = \mathbf{p}_d^{(k)} \cdot \mathbf{x}, \tag{5.18}$$

where

$$\mathbf{p}_d^{(k)} = \sum_{i=d}^{D} \mathcal{F}^i \bigotimes_{j=1}^{d-1} \mathbf{x}_j^{(k+1)} \bigotimes_{l=d+1}^{i} \mathbf{x}_l^{(k)}. \tag{5.19}$$

Now, let $\mathrm{cst}(\mathbf{x})$ denote a term that does not depend on $\mathbf{x}$ and define

$$\mathbf{s}_d^{(k)} = \sum_{i=1}^{d-1} \mathbf{A}_i \mathbf{x}_i^{(k+1)} + \sum_{j=d+1}^{D} \mathbf{A}_j \mathbf{x}_j^{(k)}. \tag{5.20}$$

The augmented Lagrangian (5.11) becomes

$$L_\rho(\mathbf{x}_{[1,d-1]}^{(k+1)}, \mathbf{x}, \mathbf{x}_{[d+1,D]}^{(k)}, \mathbf{y}^{(k)}) = \mathbf{p}_d^{(k)} \cdot \mathbf{x} + \mathbf{y}^{(k)} \cdot \left(\mathbf{A}_d \mathbf{x} + \mathbf{s}_d^{(k)} \right) + \frac{\rho}{2} \left\| \mathbf{A}_d \mathbf{x} + \mathbf{s}_d^{(k)} \right\|_2^2 + \mathrm{cst}(\mathbf{x}). \tag{5.21}$$

Therefore, it is straightforward to see that the subproblems (5.12) are reduced to minimizing quadratic functions over convex sets:

$$\mathbf{x}_d^{(k+1)} = \underset{\mathbf{x} \in \mathcal{X}_d}{\operatorname{argmin}} \left\{ \frac{1}{2} \mathbf{x}^\top \mathbf{A}_d^\top \mathbf{A}_d \mathbf{x} + \left(\mathbf{A}_d^\top \mathbf{s}_d^{(k)} + \frac{1}{\rho} (\mathbf{A}_d^\top \mathbf{y}^{(k)} + \mathbf{p}_d^{(k)}) \right) \cdot \mathbf{x} \right\}. \tag{5.22}$$

The resulted algorithm is summarized in Algorithm 5.1. We should note that this algorithm is very general and can have an infinite number of instantiations. Indeed, each choice of $(\mathbf{A}_d)_{1 \le d \le D}$ and $(\mathcal{X}_d)_{1 \le d \le D}$ in (5.8) and (5.9) — called a *decomposition* — leads to a different algorithm. The only condition for a decomposition to be valid is that the following equivalence holds:

$$\left. \begin{array}{r} \mathbf{A}_1 \mathbf{x}_1 + \mathbf{A}_2 \mathbf{x}_2 + \cdots + \mathbf{A}_D \mathbf{x}_D = \mathbf{0} \\ \mathbf{x}_d \in \mathcal{X}_d \ \forall \ 1 \le d \le D \end{array} \right\} \Longleftrightarrow \mathbf{x}_1 = \mathbf{x}_2 = \cdots = \mathbf{x}_D \in \mathcal{X}. \tag{5.23}$$

For example, if $D = 3$ then the following decomposition is valid:

$$\mathbf{x}_1 = \frac{1}{2}(\mathbf{x}_2 + \mathbf{x}_3) \tag{5.24}$$

$$\mathbf{x}_2 = \frac{1}{3}(\mathbf{x}_1 + 2\mathbf{x}_3) \tag{5.25}$$

$$\mathbf{x}_1 \in \mathcal{X}, \mathbf{x}_2 \in \mathcal{X}, \mathbf{x}_3 \in \mathcal{X}, \tag{5.26}$$

which corresponds to

$$\mathbf{A}_1 = \begin{bmatrix} \mathbf{I} \\ -\frac{1}{3}\mathbf{I} \end{bmatrix}, \quad \mathbf{A}_2 = \begin{bmatrix} -\frac{1}{3}\mathbf{I} \\ \mathbf{I} \end{bmatrix}, \quad \mathbf{A}_3 = \begin{bmatrix} -\frac{1}{2}\mathbf{I} \\ -\frac{2}{3}\mathbf{I} \end{bmatrix}. \tag{5.27}$$

With suitable choices of $(\mathbf{A}_d)_{1\le d\le D}$ and $(\mathcal{X}_d)_{1\le d\le D}$, one can obtain very simple and efficient instantiations of this algorithm. An in-depth analysis of different decompositions would be an interesting direction for future work. In the scope of this chapter, we present, analyze, and evaluate two such instantiations.

ALGORITHM 5.1 General ADGM algorithm for solving D^{th}-order graph matching.

1: Choose $(\mathbf{A}_d)_{1\le d\le D}$ and $(\mathcal{X}_d)_{1\le d\le D}$ satisfying the conditions stated in Problem 5.2.
2: Initialization: $k \leftarrow 0$, $\mathbf{y}_d^{(0)} \leftarrow \mathbf{0}$ and $\mathbf{x}_d^{(0)} \in \mathcal{X}_d$ for $d = 1, 2, \dots, D$.
3: **for** $d = 1, 2, \dots, D$ **do**
4: Compute $\mathbf{s}_d^{(k)}$ and $\mathbf{p}_d^{(k)}$ according to (5.20) and (5.19).
5: Update

$$\mathbf{x}_d^{(k+1)} \leftarrow \operatorname*{argmin}_{\mathbf{x}\in\mathcal{X}_d} \left\{ \frac{1}{2}\mathbf{x}^\top \mathbf{A}_d^\top \mathbf{A}_d \mathbf{x} + \left(\mathbf{A}_d^\top \mathbf{s}_d^{(k)} + \frac{1}{\rho}(\mathbf{A}_d^\top \mathbf{y}^{(k)} + \mathbf{p}_d^{(k)}) \right) \cdot \mathbf{x} \right\}.$$

6: **end for**
7: Update

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + \rho \left(\sum_{d=1}^{D} \mathbf{A}_d \mathbf{x}_d^{(k+1)} \right).$$

8: Compute the residual $r^{(k+1)}$ according to (5.14). If it is smaller than some threshold ϵ, then discretize $\mathbf{x}_1$ and return. Otherwise, let $k \leftarrow k + 1$ and go to Step 3.

5.3 TWO ADGM ALGORITHMS

We have introduced a general framework for solving graph matching, where the assignment vector $\mathbf{x}$ is decomposed into D vectors $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_D)$. In this section, we present two instantiations of this framework.

5.3.1 *Two simple decompositions*

First, to impose $\mathbf{x}_1 = \mathbf{x}_2 = \cdots = \mathbf{x}_D$, one can choose $(\mathbf{A}_d)_{1\le d\le D}$ such that

$$\mathbf{x}_1 = \mathbf{x}_2, \quad \mathbf{x}_1 = \mathbf{x}_3, \dots, \quad \mathbf{x}_1 = \mathbf{x}_D, \tag{5.28}$$

or alternatively

$$\mathbf{x}_1 = \mathbf{x}_2, \quad \mathbf{x}_2 = \mathbf{x}_3, \dots, \quad \mathbf{x}_{D-1} = \mathbf{x}_D. \tag{5.29}$$

It is easily seen that the above two sets of constraints can be both expressed under the general form (5.8). Each choice leads to a different algorithm. Let **ADGM1** and **ADGM2** denote the algorithms obtained from respectively (5.28) and (5.29).

To further impose that $\bigcap_{d=1}^{D} \mathcal{X}_d = \mathcal{X}$, one can make the trivial choice $\mathcal{X}_d = \mathcal{X}\ \forall d$. However, if $\mathcal{X}$ is complex, *e.g.* in the case of one-to-one matching, then ADMM subproblems (5.22) may be difficult to solve. A better choice is to use (relaxed) supersets of $\mathcal{X}$. For example, one can choose $(\mathcal{X}_d)_{1\le d\le D}$ to take values in one of the following two sets, such that *both of them are taken at least once*:

$$\mathcal{X}_{\mathrm{r}} = \{\mathbf{x} \mid \text{sum of each row of mat}(\mathbf{x}) \text{ is } \le 1\}, \tag{5.30}$$

$$\mathcal{X}_{\mathrm{c}} = \{\mathbf{x} \mid \text{sum of each column of mat}(\mathbf{x}) \text{ is } \le 1\}. \tag{5.31}$$

Again, if no occlusion is allowed then "≤ 1" is replaced by "$= 1$". If one-to-many or many-to-many matching is allowed, then these inequality constraints are removed accordingly. In either case, $\mathcal{X}_{\mathrm{r}}$ and $\mathcal{X}_{\mathrm{c}}$ are closed and convex. Clearly, since $\mathcal{X}_{\mathrm{r}} \cap \mathcal{X}_{\mathrm{c}} = \mathcal{X}$, condition (5.10) is satisfied.

5.3.2 *Update steps and resulted algorithms*

Next, we show how the subproblems (5.22) can be greatly simplified for ADGM1 and ADGM2. Indeed, plugging (5.28) and (5.29) into (5.8), we will show that (5.22) are reduced to

$$\mathbf{x}_d^{(k+1)} = \operatorname*{argmin}_{\mathbf{x}\in\mathcal{X}_d} \|\mathbf{x} - \mathbf{c}_d\|_2^2, \tag{5.32}$$

where $(\mathbf{c}_d)_{1\le d\le D}$ are defined as follows:

- For ADGM1:

$$\mathbf{c}_1 = \frac{1}{D-1}\left(\sum_{d=2}^{D}\mathbf{x}_d^{(k)} - \frac{1}{\rho}\sum_{d=2}^{D}\mathbf{y}_d^{(k)} - \frac{1}{\rho}\sum_{d=1}^{D}\mathcal{F}^d\bigotimes_{i=2}^{d}\mathbf{x}_i^{(k)}\right), \tag{5.33}$$

$$\mathbf{c}_d = \mathbf{x}_1^{(k+1)} + \frac{1}{\rho}\mathbf{y}_d^{(k)} - \frac{1}{\rho}\left(\sum_{i=d}^{D}\mathcal{F}^i\bigotimes_{i=1}^{d-1}\mathbf{x}_i^{(k+1)}\bigotimes_{j=d+1}^{i}\mathbf{x}_j^{(k)}\right), \quad 2\le d\le D, \tag{5.34}$$

- For ADGM2:

$$\mathbf{c}_1 = \mathbf{x}_2^{(k)} - \frac{1}{\rho}\mathbf{y}_2^{(k)} - \frac{1}{\rho}\sum_{d=1}^{D}\mathcal{F}^d\bigotimes_{i=2}^{d}\mathbf{x}_i^{(k)}, \tag{5.35}$$

$$\mathbf{c}_D = \mathbf{x}_{D-1}^{(k+1)} + \frac{1}{\rho}\mathbf{y}_D^{(k)} - \frac{1}{\rho}\mathcal{F}^D\bigotimes_{i=1}^{D-1}\mathbf{x}_i^{(k+1)}, \tag{5.36}$$

$$\mathbf{c}_d = \frac{1}{2}(\mathbf{x}_{d-1}^{(k+1)} + \mathbf{x}_{d+1}^{(k)}) + \frac{1}{2\rho}(\mathbf{y}_d^{(k)} - \mathbf{y}_{d+1}^{(k)}) \tag{5.37}$$

$$- \frac{1}{2\rho}\sum_{i=d}^{D}\mathcal{F}^i\bigotimes_{i=1}^{d-1}\mathbf{x}_i^{(k+1)}\bigotimes_{j=d+1}^{i}\mathbf{x}_j^{(k)}, \quad 2\le d\le D-1. \tag{5.38}$$

In the above equations, $\mathbf{y}_d$ denotes the $(d-1)^{\text{th}}$ block of the multiplier vector $\mathbf{y}$, which will become clear in the sequel.

We detail the calculation for ADGM1 and refer the reader to Appendix A.1.1 for ADGM2. Indeed, (5.28) can be written in the following form:

$$\begin{bmatrix}\mathbf{x}_1\\ \mathbf{x}_1\\ \vdots\\ \mathbf{x}_1\end{bmatrix}+\begin{bmatrix}-\mathbf{x}_2\\ \mathbf{0}\\ \vdots\\ \mathbf{0}\end{bmatrix}+\begin{bmatrix}\mathbf{0}\\ -\mathbf{x}_3\\ \vdots\\ \mathbf{0}\end{bmatrix}+\cdots+\begin{bmatrix}\mathbf{0}\\ \mathbf{0}\\ \vdots\\ -\mathbf{x}_D\end{bmatrix}=\begin{bmatrix}\mathbf{0}\\ \mathbf{0}\\ \vdots\\ \mathbf{0}\end{bmatrix}, \tag{5.39}$$

which can be in turn re-written as $\mathbf{A}_1\mathbf{x}_1+\mathbf{A}_2\mathbf{x}_2+\cdots+\mathbf{A}_D\mathbf{x}_D=\mathbf{0}$ where $\mathbf{A}_d$ is the d^{th} (block) **column** of the following $(D-1)\times D$ block matrix $\mathbf{A}$ whose blocks are $n\times n$, and as a consequence, $\mathbf{y}$ is also a $(D-1)\times 1$ block vector where each block is an n-dimensional vector:

$$\mathbf{A}=\begin{bmatrix}\mathbf{I} & -\mathbf{I} & \mathbf{0} & \cdots & \mathbf{0}\\ \mathbf{I} & \mathbf{0} & -\mathbf{I} & \cdots & \mathbf{0}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ \mathbf{I} & \mathbf{0} & \mathbf{0} & \cdots & -\mathbf{I}\end{bmatrix},\quad \mathbf{y}=\begin{bmatrix}\mathbf{y}_2\\ \mathbf{y}_3\\ \vdots\\ \mathbf{y}_D\end{bmatrix}. \tag{5.40}$$

From (5.20) we easily have

$$\mathbf{s}_1^{(k)}=\begin{bmatrix}-\mathbf{x}_2^{(k)}\\ -\mathbf{x}_3^{(k)}\\ \vdots\\ -\mathbf{x}_D^{(k)}\end{bmatrix}\quad\text{and}\quad \mathbf{s}_d^{(k)}=\begin{bmatrix}\mathbf{x}_1^{(k+1)}\\ \vdots\\ \mathbf{x}_1^{(k+1)}\\ \mathbf{x}_1^{(k+1)}\\ \mathbf{x}_1^{(k+1)}\\ \vdots\\ \mathbf{x}_1^{(k+1)}\end{bmatrix}-\begin{bmatrix}\mathbf{x}_2^{(k+1)}\\ \vdots\\ \mathbf{x}_{d-1}^{(k+1)}\\ \mathbf{0}\\ \mathbf{x}_{d+1}^{(k)}\\ \vdots\\ \mathbf{x}_D^{(k)}\end{bmatrix},\quad 2\le d\le D. \tag{5.41}$$

Next we compute the vectors $(\mathbf{c}_d)_{1\le d\le D}$.

Consider $d=1$. Since $\mathbf{A}_1=\begin{bmatrix}\mathbf{I} & \mathbf{I} & \cdots & \mathbf{I}\end{bmatrix}^\top$ we have

$$\mathbf{A}_1^\top\mathbf{A}_1=(D-1)\mathbf{I},\qquad \mathbf{A}_1^\top\mathbf{s}_1^{(k)}=-\sum_{d=2}^{D}\mathbf{x}_d^{(k)},\qquad \mathbf{A}_1^\top\mathbf{y}^{(k)}=\sum_{d=2}^{D}\mathbf{y}_d^{(k)}. \tag{5.42}$$

Plugging these into (5.22), the quadratic function therein becomes

$$\frac{1}{2}(D-1)\|\mathbf{x}\|_2^2+\left(-\sum_{d=2}^{D}\mathbf{x}_d^{(k)}+\frac{1}{\rho}\sum_{d=2}^{D}\mathbf{y}_d^{(k)}+\frac{1}{\rho}\mathbf{p}_1^{(k)}\right)\cdot\mathbf{x}. \tag{5.43}$$

Clearly, minimizing this quantity over $\mathcal{X}_1$ is equivalent to solving (5.32) for $d=1$, where $\mathbf{c}_1$ is defined by (5.33).

Now consider $d\ge 2$. Since

$$\mathbf{A}_d=\begin{bmatrix}\mathbf{0} & \cdots & \mathbf{0} & -\mathbf{I} & \mathbf{0} & \cdots & \mathbf{0}\end{bmatrix}^\top, \tag{5.44}$$

where the $-\mathbf{I}$ block is at the $(d-1)^{\text{th}}$ position, we have

$$\mathbf{A}_d^\top \mathbf{A}_d = \mathbf{I}, \qquad \mathbf{A}_d^\top \mathbf{s}_d^{(k)} = -\mathbf{x}_1^{(k+1)}, \qquad \mathbf{A}_d^\top \mathbf{y}^{(k)} = -\mathbf{y}_d^{(k)}. \tag{5.45}$$

Plugging these into (5.22), it becomes

$$\frac{1}{2}\|\mathbf{x}\|_2^2 + \left(-\mathbf{x}_1^{(k+1)} - \frac{1}{\rho}\mathbf{y}_d^{(k)} + \frac{1}{\rho}\mathbf{p}_d^{(k)}\right) \cdot \mathbf{x}. \tag{5.46}$$

Minimizing this quantity over $\mathcal{X}_d$ is equivalent to solving (5.32), where $\mathbf{c}_d$ is defined by (5.34).

We have showed that the $\mathbf{x}$-update steps (5.12) (or equivalently (5.22)) are reduced to the projections (5.32), where $(\mathbf{c}_d)_{1\le d\le D}$ are defined by (5.33)–(5.34) for ADGM1 and by (5.35)–(5.38) for ADGM2. For the $\mathbf{y}$-update (5.13), it can be seen from (5.28) and (5.29) that this step is reduced to:

$$\mathbf{y}_d^{(k+1)} = \mathbf{y}_d^{(k)} + \rho\left(\mathbf{x}_1^{(k+1)} - \mathbf{x}_d^{(k+1)}\right) \qquad \text{for ADGM1,} \tag{5.47}$$

$$\mathbf{y}_d^{(k+1)} = \mathbf{y}_d^{(k)} + \rho\left(\mathbf{x}_{d-1}^{(k+1)} - \mathbf{x}_d^{(k+1)}\right) \qquad \text{for ADGM2.} \tag{5.48}$$

The residual (5.14) is also given accordingly:

$$r^{(k+1)} = \sum_{d=2}^{D}\left\|\mathbf{x}_1^{(k+1)} - \mathbf{x}_d^{(k+1)}\right\|_2^2 + \sum_{d=1}^{D}\left\|\mathbf{x}_d^{(k+1)} - \mathbf{x}_d^{(k)}\right\|_2^2 \qquad \text{for ADGM1,} \tag{5.49}$$

$$r^{(k+1)} = \sum_{d=2}^{D}\left\|\mathbf{x}_d^{(k+1)} - \mathbf{x}_{d-1}^{(k+1)}\right\|_2^2 + \sum_{d=1}^{D}\left\|\mathbf{x}_d^{(k+1)} - \mathbf{x}_d^{(k)}\right\|_2^2 \qquad \text{for ADGM2.} \tag{5.50}$$

The two resulted algorithms are summarized in Algorithm 5.2.

ALGORITHM 5.2 Instantiations of ADGM for solving D^{th}-order graph matching.

1: Choose $(\mathcal{X}_d)_{1\le d\le D}$. For one-to-one matching these can take values in $\{\mathcal{X}_{\text{r}}, \mathcal{X}_{\text{c}}\}$, defined by (5.30) and (5.31), such that both $\mathcal{X}_{\text{r}}$ and $\mathcal{X}_{\text{c}}$ are taken at least once.
2: Initialization: $k \leftarrow 0$, $\mathbf{y}_d^{(0)} \leftarrow \mathbf{0}$ and $\mathbf{x}_d^{(0)} \in \mathcal{X}_d$ for $d = 1, \dots, D$.
3: **for** $d = 1, 2, \dots, D$ **do**
4: Compute $\mathbf{c}_d$ according to (5.33)–(5.38).
5: Update $\mathbf{x}_d^{(k+1)} \leftarrow \operatorname{argmin}_{\mathbf{x}\in\mathcal{X}_d} \|\mathbf{x} - \mathbf{c}_d\|_2^2$.
6: **end for**
7: **for** $d = 2, 3, \dots, D$ **do**
8: Update $\mathbf{y}_d^{(k+1)}$ according to (5.47)–(5.48).
9: **end for**
10: Compute the residual $r^{(k+1)}$ according to (5.49)–(5.50). If it is smaller than some threshold ϵ, then discretize $\mathbf{x}_1$ and return. Otherwise, let $k \leftarrow k+1$ and go to Step 3.

Remark. When $D = 2$ the two algorithms are identical.

5.3.3 More details on solving the subproblems

We have seen how the subproblems in the two presented ADGM algorithms can be reduced to the projections (5.32). We haven't seen, however, how to solve these projections. Recall that $\mathcal{X}_d$ is equal to either $\mathcal{X}_\mathrm{r}$ or $\mathcal{X}_\mathrm{c}$, defined by (5.30) and (5.31), *i.e.* either the sum of each row of $\mathbf{x}$ is ≤ 1, or the sum of each column of $\mathbf{x}$ is ≤ 1 (or "$= 1$" in case of no occlusion). Therefore, the above projections are reduced to *independent* projections of each row or column of $\mathbf{x}$, which can be solved using the following lemma.

Lemma 5.1. *Let d be a positive integer and $\mathbf{c} = (c_1, c_2, \dots, c_d)$ be a real-valued constant vector. Consider the problem of minimizing*

$$\|\mathbf{u} - \mathbf{c}\|_2^2 \tag{5.51}$$

with respect to $\mathbf{u} \in \mathbb{R}^d$, subject to one of the following sets of constraints:

(a) $\mathbf{u} \geq \mathbf{0}$ *and* $\mathbf{1}^\top \mathbf{u} = 1$.

(b) $\mathbf{u} \geq \mathbf{0}$ *and* $\mathbf{1}^\top \mathbf{u} \leq 1$.

An optimal solution $\mathbf{u}^*$ to each of the above two cases is given as follows:

(a) Let $\mathbf{a} = (a_1, a_2, \dots, a_d)$ be a decreasing permutation of $\mathbf{c}$ via a permutation function σ, i.e. *$a_i = c_{\sigma(i)}$ and $a_1 \geq a_2 \geq \cdots \geq a_d$. Denote*

$$\lambda_k = \frac{1}{k}\left(\sum_{1 \leq i \leq k} a_i - 1\right) \quad \forall k \in \mathbb{Z}, 1 \leq k \leq d. \tag{5.52}$$

Then there exists $k^* \in \mathbb{Z}, 1 \leq k^* \leq d$, such that $a_{k^*} > \lambda_{k^*} \geq a_{k^*+1}$. An optimal solution $\mathbf{u}^* = (u_1^*, u_2^*, \dots, u_d^*)$ is given by:

$$u^*_{\sigma(i)} = \begin{cases} a_i - \lambda_{k^*} & \text{if } 1 \leq i \leq k^*, \\ 0 & \text{if } k^* < i \leq d. \end{cases} \tag{5.53}$$

(b) Let $\mathbf{u}_0 = \max(\mathbf{c}, \mathbf{0})$. We have:

- *If $\mathbf{1}^\top \mathbf{u}_0 \leq 1$ then $\mathbf{u}^* = \mathbf{u}_0$.*
- *Otherwise, any optimal solution $\mathbf{u}^*$ must satisfy $\mathbf{1}^\top \mathbf{u}^* = 1$. Thus, the problem is reduced to the previous case, and as a consequence, $\mathbf{u}^*$ is given by (5.53).*

Part (a) is the well-known projection onto a simplex, and part (b) can be easily reduced to part (a). A proof of this lemma can be found in Appendix A.1.2. In our implementation, we used the algorithm introduced in [Condat, 2016] for this simplex projection task.

5.3.4 ADGM for solving the linear assignment problem

Recall that ADGM1 imposes $\mathbf{x}_1 = \mathbf{x}_d \ \forall\ 2 \leq d \leq D$ and ADGM2 imposes $\mathbf{x}_{d-1} = \mathbf{x}_d \ \forall\ 2 \leq d \leq D$. Clearly, these constraints are only valid for $D \geq 2$ and when $D = 2$

these two sets of constraints become the same, *i.e.* ADGM1 and ADGM2 are identical. For completeness, we briefly consider the case $D = 1$, which is the well-known linear assignment problem.

This problem can be seen as a special case of pairwise graph matching where the pairwise potentials are zeros. It can be reformulated as minimizing $F^1(\mathbf{x}_1)$ subject to $\mathbf{x}_1 = \mathbf{x}_2$ and $\mathbf{x}_1 \in \mathcal{X}_1, \mathbf{x}_2 \in \mathcal{X}_2$ (we can choose for example $\mathcal{X}_1 = \mathcal{X}_\mathrm{r}$ and $\mathcal{X}_2 = \mathcal{X}_\mathrm{c}$). Since the objective function is convex and separable, ADGM is guaranteed to produce a global optimum to the continuous relaxation of the matching problem (*c.f.* Chapter 4). However, it is well-known that this continuous relaxation is just equivalent to the original discrete problem (see *e.g.* [Schrijver, 2002, Chapter 18]). Therefore, we conclude that ADGM also produces a global optimum to the linear assignment problem.

5.3.5 *Convergent ADGM with adaptive penalty*

Note that the objective function in Problem 5.2 is neither separable nor convex in general. Convergence of ADMM for this type of functions is unknown (*c.f.* Chapter 4). Indeed, our algorithms do not always converge, especially for small values of the penalty parameter ρ (an example is given in Figure 5.1).

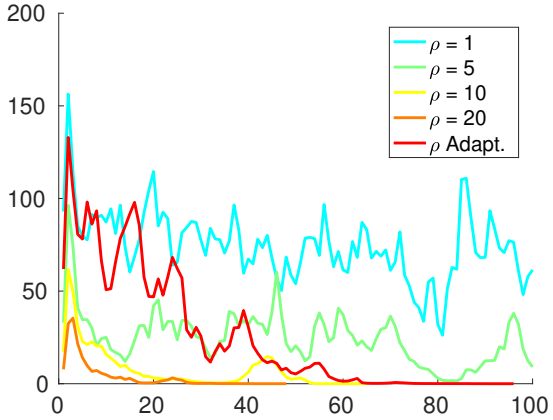


FIGURE 5.1 The residual $r^{(k)}$ per iteration of ADGM. Adaptive parameters: $T_1 = 20, T_2 = 5, \beta = \rho_0 = 2.0$ (*c.f.* Section 5.3.5). Run on a third-order Motorbike matching with 15 outliers (*c.f.* Section 5.4.2 for data and model description).

We observed that when ρ is large, ADGM algorithms are likely to converge. However, we also noticed that small ρ often (but not always) achieves better objective values. Motivated by these observations, we propose the following adaptive scheme that we find to work very well in practice:

1. Starting from a small initial value ρ_0, the algorithm runs for T_1 iterations to stabilize.
2. After that, if no improvement of the residual $r^{(k)}$ is made every T_2 iterations, then we increase ρ by a factor β and continue.

The intuition behind this scheme is simple: we hope to reach a good objective value with a small ρ, but if this leads to slow (or no) convergence, then we increase ρ to put

more penalty on the consensus of the decomposed variables and that would result in faster convergence.

Using this scheme, we observe that our algorithms always converge in practice. In the experiments, we set $T_1 = 300, T_2 = 50, \beta = 2$ and $\rho_0 = \frac{n}{1000}$.

5.4 EXPERIMENTS

We adopt the adaptive scheme presented in Section 5.3.5 to the ADGM1 and ADGM2 algorithms presented in Section 5.3. In pairwise settings, however, since these two algorithms are identical, we denote them simply ADGM. We compare **ADGM** and **ADGM1/ADGM2** to the following state-of-the-art methods.

Pairwise methods:

- Spectral Matching with Affine Constraint (**SMAC**) [Cour et al., 2007].
- Integer Projected Fixed Point (**IPFP**) [Leordeanu et al., 2009].
- Reweighted Random Walk Matching (**RRWM**) [Cho et al., 2010].
- Dual Decomposition (**DD**) [Torresani et al., 2013].
- Max-Pooling Matching (**MPM**) [Cho et al., 2014].

Higher-order methods:

- Probabilistic Graph Matching (**PGM**) [Zass and Shashua, 2008].
- Tensor Matching (**TM**) [Duchenne et al., 2011].
- Reweighted Random Walk Hypergraph Matching (**RRWHM**) [Lee et al., 2011].
- Block Coordinate Ascent Graph Matching (**BCAGM**) [Nguyen et al., 2015].

We should note that DD is only used in the experiments using the same energy models presented in [Torresani et al., 2013]. For the other experiments, DD is excluded due to the prohibitive execution time. In addition, as suggested in [Leordeanu et al., 2009], we use the solution returned by Spectral Matching (SM) [Leordeanu and Hebert, 2005] as initialization for IPFP. For BCAGM, we use MPM as subroutine because it was shown in [Nguyen et al., 2015] (and again by our experiments) that this variant of BCAGM (denoted by "BCAGM+MP" in [Nguyen et al., 2015]) outperforms the other variants thanks to the effectiveness of MPM. Since there is no ambiguity, in the sequel we denote this variant "BCAGM" for short.

We should also note that, while we formulated the graph matching as a *minimization* problem, most of the above listed methods are *maximization* solvers and many models/objective functions in previous work were designed to be maximized. For ease of comparison, ADGM is also converted to a maximization solver by letting it minimize the additive inverse of the objective function, and **the results reported in this section are for the maximization setting** (*i.e.* higher objective values are better).

In addition, for ease of comparison across different models, in the plots we show a *normalized quantity of the objective value,* which is the ratio between it and the objective value of the ground-truth matching. Furthermore, in some experiments we also use pairwise minimization models, such as the one from [Torresani et al., 2013], which we convert to maximization problems as follows: after building the affinity matrix $\mathbf{M}$ from the (minimization) potentials, the new (maximization) affinity matrix is computed by $\max(\mathbf{M})-\mathbf{M}$ where $\max(\mathbf{M})$ denotes the greatest element of $\mathbf{M}$. Note that one cannot simply take $-\mathbf{M}$ because some of the methods only work for non-negative potentials.

We present experimental results on two commonly used datasets. For each dataset, we evaluate the methods on different well-defined energy models. For a fair comparison, most of the experiments are reproduced from previous work, and we only introduce a new model when existing ones produce unsatisfactory results. Finally, to keep the presentation clear we only show the most representative results and leave additional ones to Appendix A.2.

5.4.1 House and Hotel dataset

The CMU House and Hotel sequences[1] have been widely used in previous work for evaluating graph matching algorithms. It consists of 111 frames of a synthetic house and 101 frames of a synthetic hotel. Each frame in these sequences is manually labeled with 30 feature points.

Pairwise Model A

In this experiment we match all possible pairs of images in each sequence, with all 30 points (*i.e.* no outlier). A Delaunay triangulation is performed for these 30 points to obtain the graph edges. The unary terms are the distances between the Shape Context descriptors [Belongie et al., 2002]. The pairwise terms when matching (i_1, j_1) to (i_2, j_2) are

$$\mathcal{F}^2_{ij} = \eta \exp\left(\delta^2/\sigma_l^2\right) + (1-\eta)\exp\left(\alpha^2/\sigma_a^2\right) - 1 \tag{5.54}$$

where η, σ_l, σ_a are some weight and scaling constants and δ, α are computed from $d_1 = \|\overrightarrow{i_1 j_1}\|$ and $d_2 = \|\overrightarrow{i_2 j_2}\|$ as

$$\delta = \frac{|d_1 - d_2|}{d_1 + d_2}, \quad \alpha = \arccos\left(\frac{\overrightarrow{i_1 j_1}}{d_1} \cdot \frac{\overrightarrow{i_2 j_2}}{d_2}\right). \tag{5.55}$$

This experiment is reproduced from [Torresani et al., 2013] using their publicly available energy model files[2]. It should be noted that in this model, the unary potentials are subtracted by a large number to prevent occlusion. We refer the reader to [Torresani et al., 2013] for further details.

For ease of comparison with the results reported in [Torresani et al., 2013], here we also report the performance of each algorithm in terms of overall percentage of mismatches and frequency of reaching the global optimum. Results are given in Table 5.1. One can observe that DD and ADGM always reached the global optima, but ADGM did it hundreds times faster. Even the recent methods RRWM and MPM performed

[1] http://vasc.ri.cmu.edu/idb/html/motion/index.html

[2] http://www.cs.dartmouth.edu/~lorenzo/Papers/tkr_pami13_data.zip.

TABLE 5.1 Results on the **House and Hotel** sequences using **Pairwise Model A**.

	Methods	Error (%)	Global optimum (%)	Time (s)
House	MPM	42.32	0	0.02
	RRWM	90.51	0	0.01
	IPFP	87.30	0	0.02
	SMAC	81.11	0	0.18
	DD	**0**	**100**	14.20
	ADGM	**0**	**100**	0.03
Hotel	MPM	21.49	44.80	0.02
	RRWM	85.05	0	0.01
	IPFP	85.37	0	0.02
	SMAC	71.33	0	0.18
	DD	**0.19**	**100**	13.57
	ADGM	**0.19**	**100**	0.02

poorly on this model (only MPM produced acceptable results). Also, we notice a dramatic decrease in performance of SMAC and IPFP compared to the results reported in [Torresani et al., 2013]. We should note that the above potentials, containing both positive and negative values, are defined for a *minimization* problem. It was unclear how those *maximization* solvers were used in [Torresani et al., 2013]. For the reader to be able to reproduce the results, we make our software publicly available.

Pairwise Model B

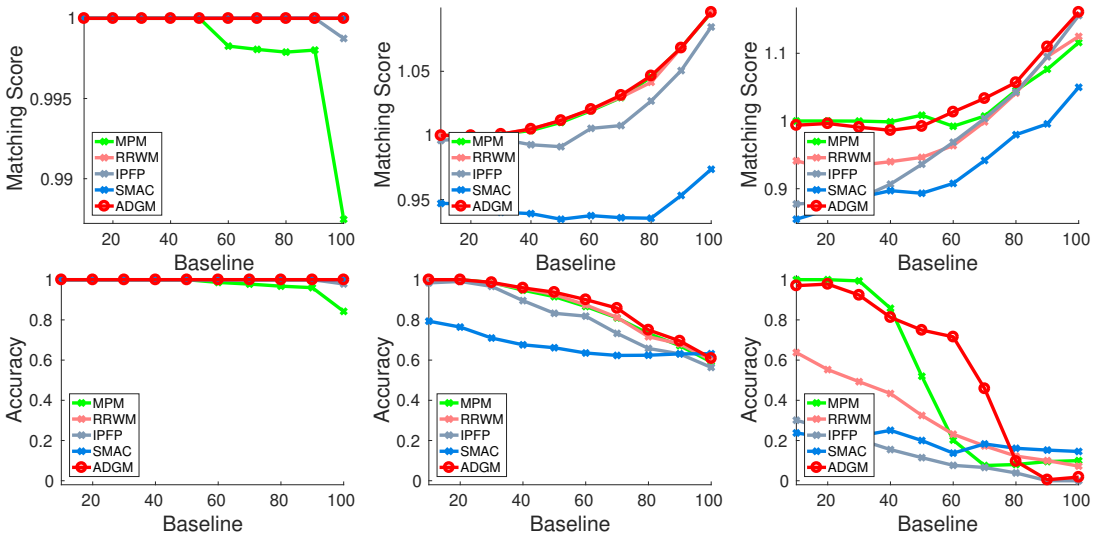
In this experiment, we match all possible pairs of the sequence with the baseline (*i.e.* the separation between frames, *e.g.* the baseline between frame 5 and frame 105 is 100) varying from 10 to 100 by intervals of 10. For each pair, we match $10, 20$ and 30 points in the first image to 30 points in the second image. We set the unary terms to 0 and compute the pairwise terms as

$$\mathcal{F}^2_{ij} = \exp\left(-\left|\|\overrightarrow{i_1 j_1}\| - \|\overrightarrow{i_2 j_2}\|\right|/\sigma^2\right), \tag{5.56}$$

where $\sigma^2 = 2500$. It should be noted that the above pairwise terms are computed for every pair (i_1, j_1) and (i_2, j_2), *i.e.* the graphs are fully connected. This experiment has been performed on the House sequence in previous work, including [Cho et al., 2010] and [Nguyen et al., 2015].

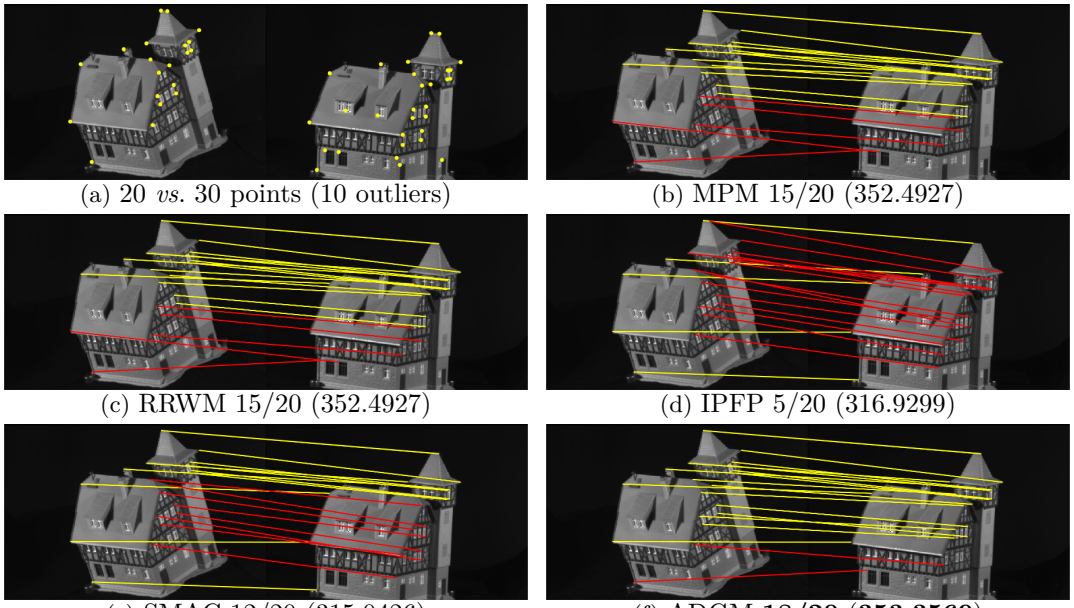
We report the averaged normalized objective value (*i.e.* matching score) and the averaged accuracy for each algorithm in Figure 5.2 for the House sequence. Qualitative results are shown in Figure 5.3. Overall, one can observe that ADGM performed best in terms of both objective value and accuracy.

We also performed the same experiments for the Hotel sequence and observed similar results. The reader is referred to Appendix A.2.1, Figure A.1 for more details, including the running time for each algorithm.



(a) 30 *vs.* 30 points (b) 20 *vs.* 30 points (c) 10 *vs.* 30 points

FIGURE 5.2 Results on the **House** sequence using **Pairwise Model B**.



(a) 20 *vs.* 30 points (10 outliers) (b) MPM 15/20 (352.4927)

(c) RRWM 15/20 (352.4927) (d) IPFP 5/20 (316.9299)

(e) SMAC 12/20 (315.0426) (f) ADGM **18/20** (**353.3569**)

FIGURE 5.3 Qualitative results on the **House** sequence using **Pairwise Model B**. The number of correct matches and the objective values are displayed. Ground-truth objective value is 343.1515. (Best viewed in color.)

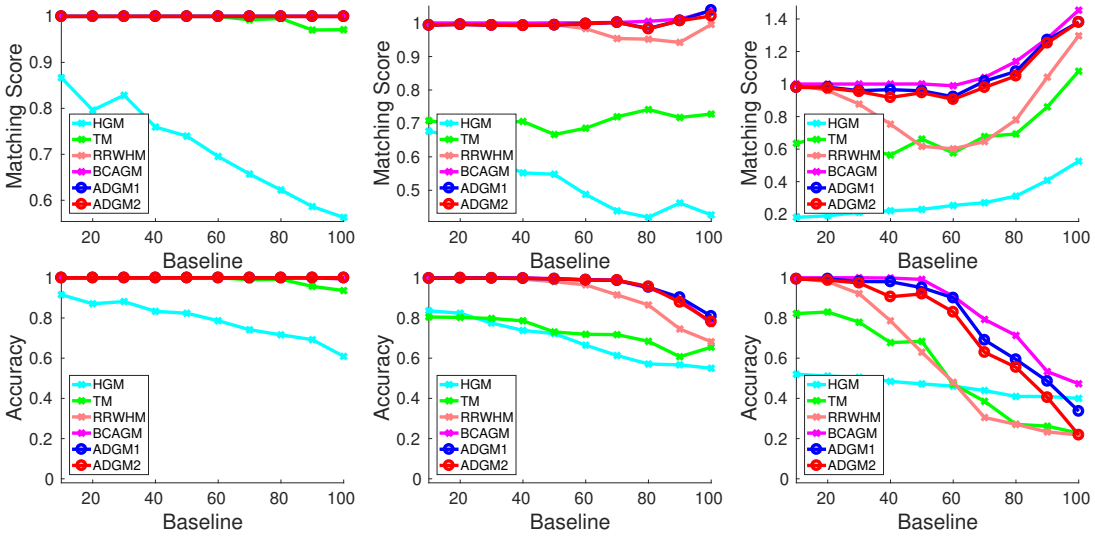
Third-order Model

This experiment has the same settings as the previous one, but here we use a third-order model. We set the unary and pairwise terms to 0 and compute the potentials when matching two triples of points (i_1, j_1, k_1) and (i_2, j_2, k_2) as

$$\mathcal{F}^3_{ijk} = \exp\left(-\left\|f_{i_1 j_1 k_1} - f_{i_2 j_2 k_2}\right\|_2^2 / \gamma\right), \tag{5.57}$$

where f_{ijk} is a feature vector composed of the angles of the triangle (i, j, k), and γ is the mean of all squared distances. This model was proposed in [Duchenne et al., 2011].

We report the results for the House sequence in Figure 5.4, and refer the reader to Appendix A.2.1 for the Hotel sequence. One can observe that ADGM1 and ADGM2 achieved quite similar performance, both were competitive with BCAGM while outperformed all the other methods.



(a) 30 *vs.* 30 points (b) 20 *vs.* 30 points (c) 10 *vs.* 30 points

FIGURE 5.4 Results on the **House** sequence using **Third-order Model**.

5.4.2 *Cars and Motorbikes dataset*

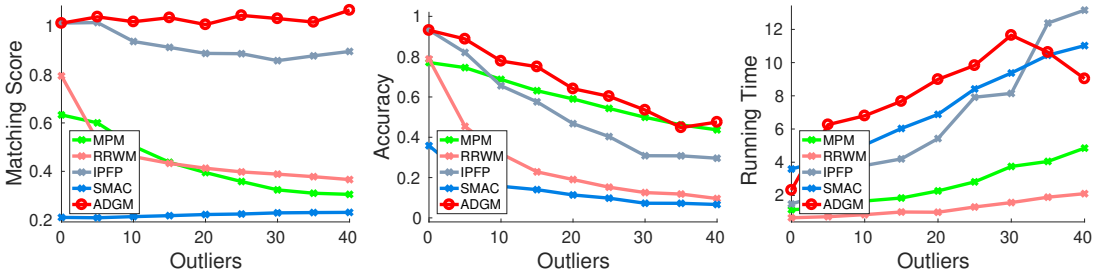
The Cars and Motorbikes dataset was introduced in [Leordeanu et al., 2012] and has been used in previous work for evaluating graph matching algorithms. It consists of 30 pairs of car images and 20 pairs of motorbike images with different shapes, view-points and scales. Each pair contains both inliers (chosen manually) and outliers (chosen randomly). For each pair of images in this dataset, we keep all inliers in both images and randomly add outliers to the second image.

The first experiment that we performed was applying Pairwise Model B (*c.f.* Section 5.4.1). However, we obtained unsatisfactory matching results, as shown in Appendix A.2.2, Figure A.7. Therefore, inspired by the model in [Torresani et al., 2013], we propose below a new model that is very simple yet very suited to matching real-world images.

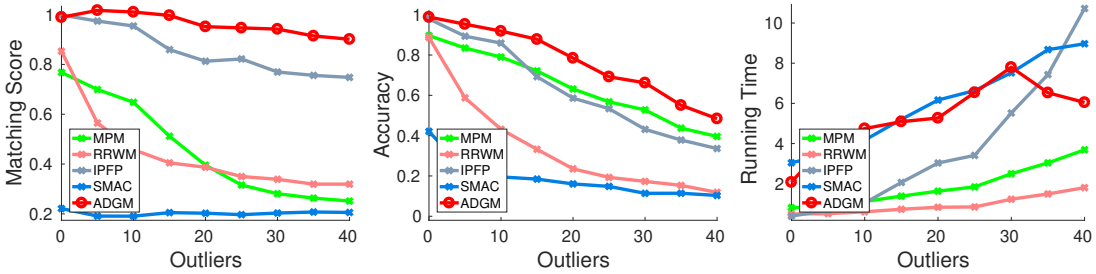
Pairwise Model C

We set the unary terms to 0 and compute the pairwise terms as

$$\mathcal{F}^2_{ij} = \eta\delta + (1-\eta)\frac{1-\cos\alpha}{2}, \tag{5.58}$$



(a) Cars



(b) Motorbikes

FIGURE 5.5 Results on the **Cars and Motorbikes** dataset using **Pairwise Model C**.

where $\eta \in [0, 1]$ is a weight constant and δ, α are computed from $d_1 = \|\overrightarrow{i_1 j_1}\|$ and $d_2 = \|\overrightarrow{i_2 j_2}\|$ as

$$\delta = \frac{|d_1 - d_2|}{d_1 + d_2}, \quad \cos\alpha = \frac{\overrightarrow{i_1 j_1}}{d_1} \cdot \frac{\overrightarrow{i_2 j_2}}{d_2}. \tag{5.59}$$

Intuitively, $\mathcal{F}^2_{ij}$ computes the geometric agreement between $\overrightarrow{i_1 j_1}$ and $\overrightarrow{i_2 j_2}$, in terms of both length and direction.

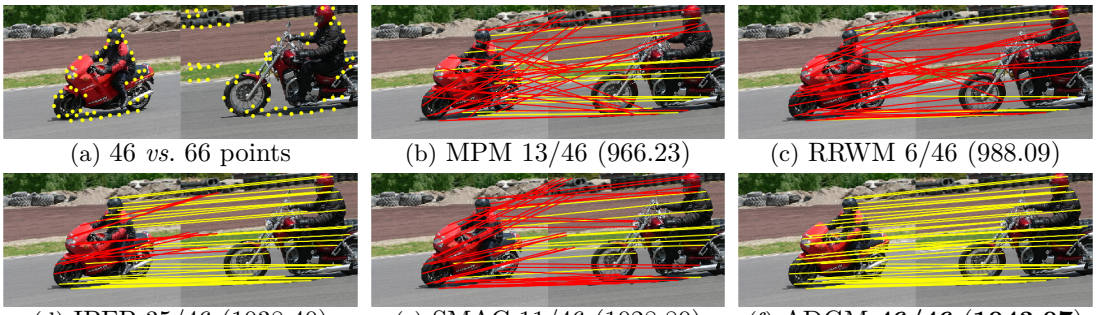
The above potentials measure the *dissimilarity* between the edges, as thus the corresponding graph matching problem is a *minimization* one. Pairwise potentials based on both length and angle were previously proposed in [Leordeanu et al., 2012, Torresani et al., 2013] and [Zhou and De la Torre, 2012]. However, ours are the simplest to compute. In this experiment, we set $\eta = 0.5$.

As we observed that this model is quite robust to outliers, we allowed the number of outliers to vary from 0 to 40 for every image pair. We report the average objective value and average matching accuracy for each method in Figure 5.5. Qualitative results are also given in Figure 5.6. One can observe that ADGM completely outperformed all the other methods.

Third-order Model

We use the same third-order model as in the House and Hotel experiments, and we allow the number of outliers to vary from 0 to 16, by intervals of 2.

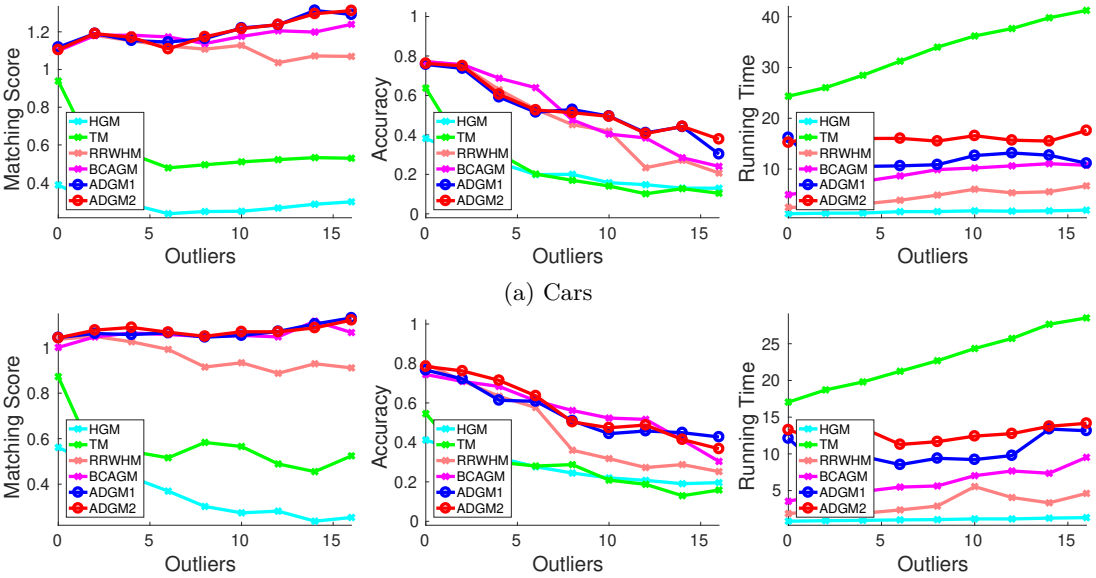
Quantitative results are reported in Figure 5.7 and qualitative results are given in Figure 5.8. ADGM performed also very well on this dataset. On Cars, both ADGM1 and ADGM2 achieved better objective values than BCAGM in 7/9 cases. On Motorbikes, ADGM1 beat BCAGM in 5/9 cases and had equivalent performance in 1/9 cases; ADGM2 beat BCAGM in 8/9 cases. Overall, one can conclude that ADGM



(a) 46 *vs.* 66 points (b) MPM 13/46 (966.23) (c) RRWM 6/46 (988.09)

(d) IPFP 35/46 (1038.40) (e) SMAC 11/46 (1028.80) (f) ADGM **46/46** (**1043.07**)

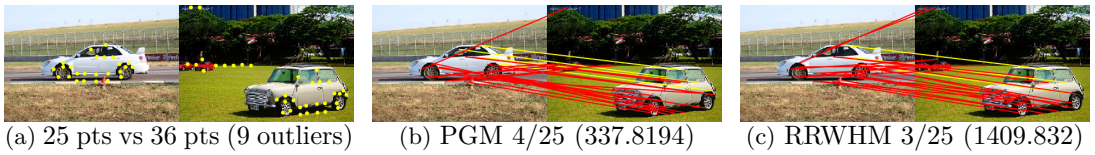
FIGURE 5.6 Qualitative results on **Motorbikes** using **Pairwise Model C**. The number of correct matches and the objective values are displayed. Ground-truth objective value is 1043.07. (Best viewed in color.)



(a) Cars

(b) Motorbikes

FIGURE 5.7 Results on the **Cars and Motorbikes** dataset using **Third-order Model**.



(a) 25 pts vs 36 pts (9 outliers) (b) PGM 4/25 (337.8194) (c) RRWHM 3/25 (1409.832)

(d) BCAGM 15/25 (1713.487) (e) ADGM1 **25/25** (**2161.5354**) (f) ADGM2 **25/25** (**2161.5354**)

FIGURE 5.8 Qualitative results on **Cars** using **Third-order Model**. The number of correct matches and the objective values are displayed. Ground-truth objective value is 2161.5354. (Best viewed in color.)

algorithms produced the best performance.

5.5 CONCLUSION

We have presented Alternating Direction Graph Matching (ADGM), a general decomposition framework for solving graph and hypergraph matching. This framework is very flexible, includes an infinite number of particular cases, and can be applied to models of arbitrary order with arbitrary potentials. Two examples of ADGM were implemented and evaluated. The results demonstrated that they outperform existing pairwise methods and competitive with the state-of-the-art higher-order methods.

6

Nonconvex Continuous Relaxation of MAP Inference

In this chapter, we study a nonconvex continuous relaxation of MAP inference in discrete Markov random fields (MRFs). We show that for arbitrary MRFs, this relaxation is tight, and a discrete stationary point of it can be easily reached by a simple block coordinate descent algorithm. In addition, we study the resolution of this relaxation using popular gradient methods, and further propose a more effective solution using a multilinear decomposition framework based on the alternating direction method of multipliers (ADMM). Experiments on many real-world problems demonstrate that the proposed ADMM significantly outperforms other nonconvex relaxation based methods, and compares favorably with state-of-the-art MRF optimization algorithms in different settings. A preliminary version of this work was published in [Lê-Huu and Paragios, 2018].

6.1 INTRODUCTION

We have seen in Chapter 2 that MAP inference methods can be grouped into 4 classes: *message passing*, *move making*, *combinatorial*, and *convex relaxation*. Viewing from a higher level, these methods can also be grouped into two bigger classes: (a) methods that stay in the discrete domain, or (b) methods that move into the continuous domain by solving convex relaxations.

While convex relaxations allow us to benefit from the tremendous convex optimization literature, and can be solved exactly in polynomial time, they often only produce real-valued solutions that need a further rounding step to be converted into integer ones, which can reduce significantly the accuracy if the relaxations are not tight. On the contrary, discrete methods tackle directly the original problem, but due to its combinatorial nature, this is a very challenging task.

In this chapter, we consider a different approach. We present a nonconvex continuous relaxation to the MAP inference problem for arbitrary (pairwise or higher-order) discrete MRFs. Based on a block coordinate descent (BCD) rounding scheme that is guaranteed not to increase the energy over continuous solutions, we show that this nonconvex relaxation is tight and is actually equivalent to the original discrete problem. It should be noted that the same relaxation was previously discussed in [Ravikumar and Lafferty, 2006] but only for *pairwise* MRFs and, more importantly, was not directly solved. The significance of this (QP) nonconvex relaxation has re-

mained purely theoretical since then. In this paper, we demonstrate it to be of great practical significance as well. In addition to establishing theoretical properties of this nonconvex relaxation for *arbitrary* MRFs based on BCD, we study popular generic optimization methods such as projected gradient descent [Bertsekas, 1999] and Frank-Wolfe algorithm [Frank and Wolfe, 1956] for solving it. These methods, however, are empirically shown to suffer greatly from the trivial hardness of nonconvex optimization: getting stuck in bad local minima. To overcome this difficulty, we propose a multilinear decomposition solultion based on the alternating direction method of multipliers (ADMM). Experiments on different real-world problems show that the proposed nonconvex based approach can outperform many of the previously mentioned methods in different settings.

The remainder of this chapter is organized as follows. First, we present the necessary mathematical notation and formulation for our approach in Section 6.2. Next, in Section 6.3, we introduce the nonconvex continuous relaxation of MAP inference, and prove that it is tight. The resolution of this relaxation using gradient methods and ADMM are presented in Section 6.4, while a theoretical convergence analysis for these methods are given in Section 6.5. Section 6.6 presents experimental validation and comparison with state-of-the-art methods. Finally, the last section concludes the chapter.

6.2 NOTATION AND PROBLEM REFORMULATION

Let $\mathcal{G}$ be a graph of n nodes with the set of cliques $\mathcal{C}$. We have seen in Chapter 2 that the MAP inference problem of an MRF that factorizes according to $\mathcal{G}$ is equivalent to minimizing the following MRF energy:

$$e(\mathbf{s}) = \sum_{C \in \mathcal{C}} f_C(\mathbf{s}_C), \tag{6.1}$$

where f_C is the log potential function of the clique $C \in \mathcal{C}$. The n underlying random variables $S_1, S_2, \dots, S_n$ are supposed to take values in finite sets of labels (or states) $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_n$, respectively. The variable $\mathbf{s}_C \in \mathcal{S}_C := \bigtimes_{i \in C} \mathcal{S}_i$ denotes a joint label assigned to the nodes in C, and the variable $\mathbf{s} \in \mathcal{S} := \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_n$ denotes a joint label assigned to all nodes. Here we have decided to use the variable name $\mathbf{s}$ instead of $\mathbf{x}$ (as in Chapter 2) because we would like to reserve $\mathbf{x}$ as the variables for our main optimization problems in this chapter.

It is often convenient to rewrite the energy (6.1) using the indicator functions of labels assigned to each node. Let $\mathcal{V} \subset \mathcal{C}$ denote the set of nodes of the graph $\mathcal{G}$. For each $i \in \mathcal{V}$, let $x_i : \mathcal{S}_i \to \{0, 1\}$ be a function defined by

$$x_i(s) = \begin{cases} 1 & \text{if the node } i \text{ takes the label } s \in \mathcal{S}_i, \\ 0 & \text{otherwise.} \end{cases} \tag{6.2}$$

It is easily seen that minimizing $e(\mathbf{s})$ over $\mathcal{S}$ is equivalent to the following problem,

where we have rewritten $e(\mathbf{s})$ as a function of $\{x_i(\cdot)\}_{i\in\mathcal{V}}$:

$$\begin{aligned} \min \quad & E(\mathbf{x}) := \sum_{C\in\mathcal{C}} \sum_{\mathbf{s}_C\in\mathcal{S}_C} f_C(\mathbf{s}_C) \prod_{i\in C} x_i(s_i), \\ \text{s.t.} \quad & \sum_{s\in\mathcal{S}_i} x_i(s) = 1 \quad \forall i \in \mathcal{V}, \\ & x_i(s) \in \{0,1\} \quad \forall s \in \mathcal{S}_i, \forall i \in \mathcal{V}. \end{aligned} \tag{6.3}$$

In the standard LP relaxation [Wainwright et al., 2005], the product $\prod_{i\in C} x_i(s_i)$ in (6.3) is replaced with new variables $x_C(\mathbf{s}_C)$, seen as the indicator function of the joint label assigned to the clique C, and the following *local consistency* constraints are added:

$$\sum_{s_{C\setminus i}} x_C(\mathbf{s}_C) = x_i(s_i) \;\; \forall i \in C, \forall s_i \in \mathcal{S}_i. \tag{6.4}$$

In this work, we consider (6.3) but under a different formulation using tensors, just for later convenience. The reader is referred to Section 3.2.1 for tensor-related notation.

For any node i, let $\mathbf{x}_i = (x_i(s))_{s\in\mathcal{S}_i}$ be the vector composed of all possible values of $x_i(s)$. For a clique $C = (i_1, i_2, \dots, i_\alpha)$, the potential function $f_C(s_1, s_2, \dots, s_\alpha)$, where $s_d \in \mathcal{S}_{i_d} \forall 1 \le d \le \alpha$, has α indices and thus can be seen as an α^{th}-order tensor of dimensions $\left|\mathcal{S}_{i_1}\right| \times \left|\mathcal{S}_{i_2}\right| \times \cdots \times \left|\mathcal{S}_{i_\alpha}\right|$. Let $\mathbf{F}_C$ denote this tensor. Recall that the energy term corresponding to C in (6.3) is

$$\sum_{s_1,s_2,\dots,s_\alpha} f_C(s_1, s_2, \dots, s_\alpha) x_{i_1}(s_1) x_{i_2}(s_2) \cdots x_{i_\alpha}(s_\alpha), \tag{6.5}$$

which is clearly $\mathbf{F}_C \bigotimes_{\{1,2,\dots,\alpha\}} \left\{\mathbf{x}_{i_1}, \mathbf{x}_{i_2}, \dots, \mathbf{x}_{i_\alpha}\right\}$. For clarity purpose, we omit the index set and write simply $\mathbf{F}_C \bigotimes \left\{\mathbf{x}_{i_1}, \mathbf{x}_{i_2}, \dots, \mathbf{x}_{i_\alpha}\right\}$, or equivalently $\mathbf{F}_C \bigotimes \{\mathbf{x}_i\}_{i\in C}$, with the assumption that each vector is multiplied at the right mode (which is the same as its position in the clique). Therefore, the energy in (6.3) becomes

$$E(\mathbf{x}) = \sum_{C\in\mathcal{C}} \mathbf{F}_C \bigotimes \{\mathbf{x}_i\}_{i\in C}\,. \tag{6.6}$$

Problem (6.3) can then be rewritten as

$$\begin{aligned} \min \quad & E(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x} \in \overline{\mathcal{X}} := \left\{\mathbf{x} \;\middle|\; \mathbf{1}^\top \mathbf{x}_i = 1, \mathbf{x}_i \in \{0,1\}^{|\mathcal{S}_i|} \;\; \forall i \in \mathcal{V}\right\}. \end{aligned} \tag{MRF}$$

In the next section, we study a continuous relaxation of this problem.

6.3 TIGHT CONTINUOUS RELAXATION OF MAP INFERENCE

By simply relaxing the constraints $\mathbf{x}_i \in \{0,1\}^{|\mathcal{S}_i|}$ in (MRF) to $\mathbf{x}_i \geq \mathbf{0}$, we obtain the following nonconvex relaxation:

$$\begin{aligned} \min \quad & E(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x} \in \mathcal{X} := \left\{ \mathbf{x} \;\middle|\; \mathbf{1}^\top \mathbf{x}_i = 1, \mathbf{x}_i \geq \mathbf{0} \;\forall i \in \mathcal{V} \right\}. \end{aligned} \tag{RLX}$$

A clear advantage of this relaxation over the LP relaxation is its compactness. Indeed, if all nodes have the same number of labels S, then the number of variables and number of constraints of this relaxation are respectively $|\mathcal{V}|S$ and $|\mathcal{V}|$, while for the LP relaxation these numbers are respectively $\mathcal{O}(|\mathcal{C}|S^D)$ and $\mathcal{O}(|\mathcal{C}|SD)$, where D is the degree of the MRF.

In this section some interesting properties of (RLX) are presented. In particular, we prove that this relaxation is tight and show how to obtain a *discrete* stationary point for it. Let us first propose a simple BCD algorithm to solve (RLX). Relaxation tightness and other properties follow naturally.

Let $n = |\mathcal{V}|$ be the number of nodes. The vector $\mathbf{x}$ can be seen as an n-block vector, where each block corresponds to each node: $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$. Starting from an initial solution, BCD solves (RLX) by iteratively optimizing E over $\mathbf{x}_i$ while fixing all the other blocks. Note that our subsequent analysis is still valid for other variants of BCD, such as updating in a random order, or using subgraphs such as trees (instead of single nodes) as update blocks. To keep the presentation simple, however, we choose to update in the deterministic order $i = 1, 2, \dots, n$. Each update step consists of solving

$$\mathbf{x}_i^{(k+1)} \in \operatorname*{argmin}_{\mathbf{1}^\top \mathbf{x}_i = 1, \mathbf{x}_i \geq \mathbf{0}} E(\mathbf{x}_{[1,i-1]}^{(k+1)}, \mathbf{x}_i, \mathbf{x}_{[i+1,n]}^{(k)}). \tag{6.7}$$

From (6.6) it is clear that for the cliques that do not contain the node i, their corresponding energy terms are independent of $\mathbf{x}_i$. Thus, if $\mathcal{C}(i)$ denotes the set of cliques containing i, then

$$E(\mathbf{x}) = \sum_{C \in \mathcal{C}(i)} \mathbf{F}_C \bigotimes \{\mathbf{x}_j\}_{j \in C} + \operatorname{cst}(\mathbf{x}_i) \tag{6.8}$$

$$= \mathbf{c}_i^\top \mathbf{x}_i + \operatorname{cst}(\mathbf{x}_i), \tag{6.9}$$

where $\operatorname{cst}(\mathbf{x}_i)$ is a term that does not depend on $\mathbf{x}_i$, and

$$\mathbf{c}_i = \sum_{C \in \mathcal{C}(i)} \mathbf{F}_C \bigotimes \{\mathbf{x}_j\}_{j \in C \setminus i} \qquad \forall i \in \mathcal{V}. \tag{6.10}$$

The update (6.7) becomes minimizing $\mathbf{c}_i^\top \mathbf{x}_i$, which can be solved using the following straightforward lemma.

Lemma 6.1. *Let* $\mathbf{c} = (c_1, \dots, c_p) \in \mathbb{R}^p$, $\alpha = \operatorname{argmin}_\beta c_\beta$. *The problem* $\min_{\mathbf{1}^\top \mathbf{u} = 1, \mathbf{u} \geq \mathbf{0}} \mathbf{c}^\top \mathbf{u}$ *has an optimal solution* $\mathbf{u}^* = (u_1^*, \dots, u_p^*)$ *defined by* $u_\alpha^* = 1$ *and* $u_\beta^* = 0$ $\forall \beta \neq \alpha$.

According to this lemma, we can solve (6.7) as follows: compute $\mathbf{c}_i$ using (6.10),

find the position s of its smallest element, set $x_i(s) = 1$ and $x_i(r) = 0\ \forall r \neq s$. Clearly, the solution $\mathbf{x}_i$ returned by this update step is discrete. It is easily seen that this update is equivalent to assigning the node i with the following label:

$$s_i = \operatorname*{argmin}_{s \in \mathcal{S}_i} \sum_{C \in \mathcal{C}(i)} \sum_{s_{C\setminus i} \in \mathcal{S}_{C\setminus i}} f_C(s_{C\setminus i}, s) \prod_{j \in C\setminus i} x_j(s_j). \tag{6.11}$$

A sketch of the BCD algorithm is given in Algorithm 6.1.

ALGORITHM 6.1 Block coordinate descent for solving (RLX).

1: Initialization: $k \leftarrow 0$, $\mathbf{x}^{(0)} \in \mathcal{X}$.
2: **for** $i = 1, 2, \ldots, n$ **do**
3: Update $\mathbf{x}_i^{(k+1)}$ as a (discrete) solution to (6.7).
4: If $\mathbf{x}_i^{(k)}$ is also a discrete solution to (6.7), then set $\mathbf{x}_i^{(k+1)} \leftarrow \mathbf{x}_i^{(k)}$.
5: **end for**
6: Let $k \leftarrow k + 1$ and go to Step 2 until $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$.

Remark. Starting from a discrete solution (or starting from the second outer iteration), BCD is equivalent to Iterated Conditional Modes (ICM) [Besag, 1986]. Note, however, that BCD is designed to solve the *continuous* problem (RLX), whereas ICM solves directly the *discrete* problem (MRF).

We have the following convergence result for BCD.

Proposition 6.1. *For any initial solution $\mathbf{x}^{(0)}$, BCD (Algorithm 6.1) converges to a discrete fixed point.*

Proof. Clearly, BCD stops when there is no *strict* descent of the energy. Since the solution at each iteration is discrete and the number of nodes as well as the number of labels are finite, BCD must stop after a finite number of iterations. Suppose that this number is k: $E(\mathbf{x}^{(k+1)}) = E(\mathbf{x}^{(k)})$. At each inner iteration (*i.e.* Step 2 in Algorithm 6.1), the label of a node is changed to a new label only if the new label can produce *strictly* lower energy. Therefore, the labeling of $\mathbf{x}^{(k+1)}$ and $\mathbf{x}^{(k)}$ must be the same because they have the same energy, which implies $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$, *i.e.* $\mathbf{x}^{(k)}$ is a fixed point. □

We will see in Section 6.5 that the fixed point produced by BCD is also a stationary point of (RLX). Next we state and prove the main result of this section.

Theorem 6.1. *The continuous relaxation (RLX) is tight.*

Proof. Since $E(\mathbf{x})$ is continuous and both $\overline{\mathcal{X}}$ and $\mathcal{X}$ are closed, according to the Weierstrass extreme value theorem, both (MRF) and (RLX) must attain a (global) minimum, which we denote by $\mathbf{x}_{\text{MRF}}$ and $\mathbf{x}_{\text{RLX}}$, respectively. Obviously $E(\mathbf{x}_{\text{RLX}}) \leq E(\mathbf{x}_{\text{MRF}})$. Now let $\mathbf{x}^*$ be the solution of BCD with initialization $\mathbf{x}^{(0)} = \mathbf{x}_{\text{RLX}}$. On the one hand, since BCD is a descent algorithm, we have $E(\mathbf{x}^*) \leq E(\mathbf{x}_{\text{RLX}})$. On the other hand, since the solution returned by BCD is discrete, we have $\mathbf{x}^* \in \overline{\mathcal{X}}$, yielding $E(\mathbf{x}_{\text{MRF}}) \leq E(\mathbf{x}^*)$. Putting it all together, we get $E(\mathbf{x}^*) \leq E(\mathbf{x}_{\text{RLX}}) \leq E(\mathbf{x}_{\text{MRF}}) \leq E(\mathbf{x}^*)$, which implies $E(\mathbf{x}_{\text{RLX}}) = E(\mathbf{x}_{\text{MRF}})$, *i.e.* (RLX) is tight. □

Remark. The above proof is still valid if BCD performs only the first outer iteration. This means that one can obtain $\mathbf{x}_{\text{MRF}}$ from $\mathbf{x}_{\text{RLX}}$ (both have the same energy) in polynomial time, *i.e.* (RLX) and (MRF) can be seen as equivalent. This result was previously presented in [Ravikumar and Lafferty, 2006] for pairwise MRFs, here we have extended it to arbitrary MRFs.

While BCD is guaranteed to reach a discrete stationary point of (RLX), there is no guarantee on the quality of such point. In practice, as shown later in the experiments, the performance of BCD compares poorly with state-of-the-art MRF optimization methods. In fact, the key challenge in nonconvex optimization is that there might be many local minima, and as a consequence, algorithms can easily get trapped in *bad* ones, even from multiple initializations.

In the next section, we study the resolution of (RLX) using more sophisticated methods, where we come up with a multilinear decomposition ADMM that can reach very good local minima (many times even the global ones) on different real-world models.

6.4 SOLVING THE TIGHT CONTINUOUS RELAXATION

Since the MRF energy (6.6) is differentiable, it is worth investigating whether gradient methods can effectively optimize it. We present two such methods in Section 6.4.1. Then our proposed ADMM based algorithm is presented in Section 6.4.2. We provide a convergence analysis for all methods in Section 6.5.

6.4.1 *Gradient methods*

Projected gradient descent (PGD) and Frank-Wolfe algorithm (FW) are among the most popular methods for solving constrained optimization. We refer to [Bertsekas, 1999] for an excellent presentation of these methods. Here we briefly describe how to use them to solve (RLX).

ALGORITHM 6.2 Projected gradient descent for solving (RLX).

1: Initialization: $k \leftarrow 0$, $\mathbf{x}^{(0)} \in \mathcal{X}$.
2: Compute $\beta^{(k)}$ and find the projection

$$\mathbf{s}^{(k)} \leftarrow \operatorname*{argmin}_{\mathbf{s}\in\mathcal{X}} \left\|\mathbf{x}^{(k)} - \beta^{(k)}\nabla E(\mathbf{x}^{(k)}) - \mathbf{s}\right\|_2^2. \tag{6.12}$$

3: Compute $\alpha^{(k)}$ and update

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \alpha^{(k)}(\mathbf{s}^{(k)} - \mathbf{x}^{(k)}). \tag{6.13}$$

Let $k \leftarrow k+1$ and go to Step 2.

The general steps are sketched in Algorithms 6.2 and 6.3. We discuss how to solve the subproblems (6.12) and (6.14), and how to update the step-sizes $\alpha^{(k)}$ and $\beta^{(k)}$.

ALGORITHM 6.3 Frank-Wolfe algorithm for solving (RLX).

1: Initialization: $k \leftarrow 0$, $\mathbf{x}^{(0)} \in \mathcal{X}$.
2: Find
$$\mathbf{s}^{(k)} \leftarrow \operatorname*{argmin}_{\mathbf{s}\in\mathcal{X}} \mathbf{s}^\top \nabla E(\mathbf{x}^{(k)}). \tag{6.14}$$
3: Compute $\alpha^{(k)}$ and update
$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \alpha^{(k)}(\mathbf{s}^{(k)} - \mathbf{x}^{(k)}). \tag{6.15}$$
Let $k \leftarrow k+1$ and go to Step 2.

Solving the subproblems

Clearly, in the PGD subproblem (6.12), $\mathbf{s}^{(k)}$ is the projection of $\mathbf{x}^{(k)} - \beta^{(k)}\nabla E(\mathbf{x}^{(k)})$ onto $\mathcal{X}$, defined in (RLX) as $\mathcal{X} := \left\{\mathbf{x} \;\middle|\; \mathbf{1}^\top\mathbf{x}_i = 1, \mathbf{x}_i \geq \mathbf{0}\ \forall i \in \mathcal{V}\right\}$. Since the constraint on one node is independent of another, the above projection is reduced to independent projections onto the simplex $\{\mathbf{x}_i \mid \mathbf{1}^\top\mathbf{x}_i = 1, \mathbf{x}_i \geq \mathbf{0}\}$ for each node i. We have seen in Chapter 5 (Section 5.3) how to solve these simplex projections. Again in our implementation we used the fast algorithm introduced in [Condat, 2016] for this task. As we will see later in Section 6.4.2, similar subproblems arise again when applying ADMM to solve (RLX).

Similar reasoning applies to the FW subproblem (6.14), which can also be reduced to solving independently problems on each node:

$$\mathbf{s}_i^{(k)} = \operatorname*{argmin}_{\mathbf{1}^\top\mathbf{s}_i=1, \mathbf{s}_i\geq\mathbf{0}} \mathbf{s}_i^\top \frac{\partial E(\mathbf{x}^{(k)})}{\partial \mathbf{x}_i} \quad \forall i \in \mathcal{V}, \tag{6.16}$$

The above is similar to the BCD update step (6.7) and thus can also be solved using Lemma 6.1.

Updating the step-sizes

The step-sizes $\alpha^{(k)}$ and $\beta^{(k)}$ can follow different update rules [Bertsekas, 1999]. The most straightforward is the *diminishing* rule, which has for example:

$$\alpha^{(k)} = 1, \quad \beta^{(k)} = \frac{1}{\sqrt{k+1}} \qquad \text{for PGD,} \tag{6.17}$$
$$\alpha^{(k)} = \frac{2}{k+2} \qquad \text{for FW.} \tag{6.18}$$

However, in practice, these step-sizes often lead to slow convergence.[1] A better alternative is the following *line-search*:

$$\alpha^{(k)} = \operatorname*{argmin}_{0\leq\alpha\leq 1} E\left(\mathbf{x}^{(k)} + \alpha\mathbf{r}^{(k)}\right), \tag{6.19}$$

[1] Indeed, we implemented different step-size update rules such as *diminishing* or *Armijo* ones. However, we found that these rules do not work as well as *line-search* (the diminishing rule converges slowly while the search in the Armijo rule is expensive). We refer to [Bertsekas, 1999, Chapter 2] for further details on these rules.

where $\mathbf{r}^{(k)} = \mathbf{s}^{(k)} - \mathbf{x}^{(k)}$. For PGD $\beta^{(k)}$ is implicitly set to 1.

Clearly, the term $E\left(\mathbf{x}^{(k)} + \alpha\mathbf{r}^{(k)}\right)$ is a D^{th}-degree polynomial of α (recall that D is the degree of the MRF), which we denote $p(\alpha)$. If we can determine the coefficients of $p(\alpha)$, then (6.19) can be solved efficiently. In particular, if $D \leq 3$ then (6.19) has simple closed-form solutions (since the derivative of a 3^{rd}-order polynomial is a 2^{nd}-order one, which has simple closed-form solutions). For $D > 3$ we find that it is efficient enough to perform an exhaustive search over the interval $[0, 1]$ (with some increment value δ) to find the best value of α. In our implementation we used $\delta = 0.0001$.

Now let us describe how to find the coefficients of $p(\alpha)$.

For pairwise MRFs (*i.e.* $D = 2$), the energy is

$$E_{\text{pairwise}}(\mathbf{x}) = \sum_{i\in\mathcal{V}} \mathbf{F}_i^\top \mathbf{x}_i + \sum_{ij\in\mathcal{E}} \mathbf{x}_i^\top \mathbf{F}_{ij} \mathbf{x}_j, \tag{6.20}$$

where $\mathcal{E}$ is the set of edges, and thus

$$p(\alpha) = E_{\text{pairwise}}(\mathbf{x} + \alpha\mathbf{r}) = \sum_{i\in\mathcal{V}} \mathbf{F}_i^\top (\mathbf{x}_i + \alpha\mathbf{r}_i) + \sum_{ij\in\mathcal{E}} (\mathbf{x}_i + \alpha\mathbf{r}_i)^\top \mathbf{F}_{ij} (\mathbf{x}_j + \alpha\mathbf{r}_j) \tag{6.21}$$

$$= A\alpha^2 + B\alpha + C, \tag{6.22}$$

where

$$A = \sum_{ij\in\mathcal{E}} \mathbf{r}_i^\top \mathbf{F}_{ij} \mathbf{r}_j \tag{6.23}$$

$$B = \sum_{i\in\mathcal{V}} \mathbf{F}_i^\top \mathbf{r}_i + \sum_{ij\in\mathcal{E}} \left(\mathbf{x}_i^\top \mathbf{F}_{ij} \mathbf{r}_j + \mathbf{r}_i^\top \mathbf{F}_{ij} \mathbf{x}_j\right) \tag{6.24}$$

$$C = E_{\text{pairwise}}(\mathbf{x}). \tag{6.25}$$

For higher-order MRFs, the analytical expressions of the polynomial coefficients are very complicated. Instead, we can find them numerically as follows. Since $p(\alpha)$ is a D^{th}-degree polynomial, it has $D + 1$ coefficients, where the constant coefficient is already known:

$$p(0) = E(\mathbf{x}^{(k)}). \tag{6.26}$$

It remains D unknown coefficients, which can be computed if we have D equations. Indeed, if we evaluate $p(\alpha)$ at D different random values of α (which must be different from 0), then we obtain D linear equations whose variables are the coefficients of $p(\alpha)$. Solving this system of linear equations we get the values of these coefficients. This procedure requires D evaluations of the energy $E\left(\mathbf{x}^{(k)} + \alpha\mathbf{r}^{(k)}\right)$, but we find that it is efficient enough in practice.

6.4.2 *Alternating direction method of multipliers*

In this section, we will apply ADMM to solve (RLX). The idea is the same as the ADGM algorithms presented in Chapter 5. However, to make ADMM efficient and effective for MAP inference, we add the following important practical contributions: (1) We formulate the problem using individual potential tensors at each clique (instead of a single large tensor as in ADGM), which allows a better exploitation of the

problem structure, as computational quantities at each node can be cached based on its neighboring nodes, yielding significant speed-ups; (2) We discuss how to choose the decomposed constraint sets that result in the best accuracy for MAP inference (note that the constraint sets for graph matching are different). In addition, we present a convergence analysis for the proposed method in Section 6.5.

For the reader to quickly get the idea, let us start with an example of a second-order[2] MRF:

$$E_{\text{second}}(\mathbf{x}) = \sum_{i\in\mathcal{V}} \mathbf{F}_i \bigotimes \mathbf{x}_i + \sum_{ij\in\mathcal{C}} \mathbf{F}_{ij} \bigotimes \{\mathbf{x}_i, \mathbf{x}_j\} + \sum_{ijk\in\mathcal{C}} \mathbf{F}_{ijk} \bigotimes \{\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k\}. \quad (6.27)$$

Instead of dealing directly with this high degree polynomial, which is highly challenging, the idea is to decompose $\mathbf{x}$ into different variables that can be handled separately using Lagrangian relaxation. To this end, consider the following multilinear function:

$$F_{\text{second}}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \sum_{i\in\mathcal{V}} \mathbf{F}_i \bigotimes \mathbf{x}_i + \sum_{ij\in\mathcal{C}} \mathbf{F}_{ij} \bigotimes \{\mathbf{x}_i, \mathbf{y}_j\} + \sum_{ijk\in\mathcal{C}} \mathbf{F}_{ijk} \bigotimes \{\mathbf{x}_i, \mathbf{y}_j, \mathbf{z}_k\}. \quad (6.28)$$

Clearly, $E_{\text{second}}(\mathbf{x}) = F_{\text{second}}(\mathbf{x}, \mathbf{x}, \mathbf{x})$. Thus, minimizing $E(\mathbf{x})$ is equivalent to minimizing $F_{\text{second}}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ under the constraints $\mathbf{x} = \mathbf{y} = \mathbf{z}$, which can be relaxed using Lagrangian based method such as ADMM.

Back to our general problem (RLX). Let D denote the maximum clique size of the corresponding MRF. Using the same idea as above for decomposing $\mathbf{x}$ into D vectors $\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^D$, let us define

$$F(\mathbf{x}^1, \dots, \mathbf{x}^D) = \sum_{d=1}^{D} \sum_{i_1\dots i_d\in\mathcal{C}} \mathbf{F}_{i_1\dots i_d} \bigotimes \left\{\mathbf{x}^1_{i_1}, \dots, \mathbf{x}^d_{i_d}\right\}. \quad (6.29)$$

Clearly, the energy (6.6) becomes $E(\mathbf{x}) = F(\mathbf{x}, \mathbf{x}, \dots, \mathbf{x})$. It is straightforward to see that (RLX) is equivalent to:

$$\begin{aligned} \min \quad & F(\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^D) \\ \text{s.t.} \quad & \mathbf{A}^1\mathbf{x}^1 + \dots + \mathbf{A}^D\mathbf{x}^D = \mathbf{0}, \\ & \mathbf{x}^d \in \mathcal{X}^d, \quad d = 1, \dots, D, \end{aligned} \quad (6.30)$$

where $\mathbf{A}^1, \dots, \mathbf{A}^D$ are constant matrices such that

$$\mathbf{A}^1\mathbf{x}^1 + \dots + \mathbf{A}^D\mathbf{x}^D = \mathbf{0} \Longleftrightarrow \mathbf{x}^1 = \dots = \mathbf{x}^D, \quad (6.31)$$

and $\mathcal{X}^1, \dots, \mathcal{X}^D$ are closed convex sets satisfying

$$\mathcal{X}^1 \cap \mathcal{X}^2 \cap \dots \cap \mathcal{X}^D = \mathcal{X}. \quad (6.32)$$

Note that the linear constraint in (6.30) is a general way to enforce $\mathbf{x}^1 = \dots = \mathbf{x}^D$ and it has an infinite number of particular instances. For example, with suitable choices of $(\mathbf{A}^d)_{1\le d\le D}$, this linear constraint can become either one of the following sets of

[2]Note that pairwise MRFs are also called *first-order* ones.

constraints:

$$\begin{aligned}
&\text{(cyclic)} && \mathbf{x}^{d-1} = \mathbf{x}^d, \quad d = 2, \dots, D, && (6.33)\\
&\text{(star)} && \mathbf{x}^1 = \mathbf{x}^d, \quad d = 2, \dots, D, && (6.34)\\
&\text{(symmetric)} && \mathbf{x}^d = (\mathbf{x}^1 + \cdots + \mathbf{x}^D)/D \quad \forall d. && (6.35)
\end{aligned}$$

We call such an instance a *decomposition*, and each decomposition will lead to a different algorithm.

The augmented Lagrangian of (6.30) is defined by:

$$L_\rho(\mathbf{x}^1, \dots, \mathbf{x}^D, \mathbf{y}) = F(\mathbf{x}^1, \dots, \mathbf{x}^D) + \mathbf{y}^\top \left(\sum_{d=1}^D \mathbf{A}^d \mathbf{x}^d \right) + \frac{\rho}{2} \left\| \sum_{d=1}^D \mathbf{A}^d \mathbf{x}^d \right\|_2^2. \tag{6.36}$$

Recall that standard ADMM (Chapter 4) solves (6.30) by iterating:

1. For $d = 1, 2, \dots, D$: update $\mathbf{x}^{d^{(k+1)}}$ as a solution of

$$\min_{\mathbf{x}^d \in \mathcal{X}^d} L_\rho(\mathbf{x}^{[1,d-1]^{(k+1)}}, \mathbf{x}^d, \mathbf{x}^{[d+1,D]^{(k)}}, \mathbf{y}^{(k)}). \tag{6.37}$$

2. Update $\mathbf{y}$:

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + \rho \left(\sum_{d=1}^D \mathbf{A}^d \mathbf{x}^{d^{(k+1)}} \right). \tag{6.38}$$

The algorithm converges if the following *residual* converges to 0 as $k \to +\infty$:

$$r^{(k)} = \left\| \sum_{d=1}^D \mathbf{A}^d \mathbf{x}^{d^{(k)}} \right\|_2^2 + \sum_{d=1}^D \left\| \mathbf{x}^{d^{(k)}} - \mathbf{x}^{d^{(k-1)}} \right\|_2^2. \tag{6.39}$$

We show how to solve the $\mathbf{x}$ update step (6.37) (the $\mathbf{y}$ update (6.38) is trivial). Updating $\mathbf{x}^d$ consists of minimizing the augmented Lagrangian (6.36) with respect to the d^{th} block while fixing the other blocks.

Since $F(\mathbf{x}^1, \dots, \mathbf{x}^D)$ is linear with respect to each block $\mathbf{x}^d$ (*c.f.* (6.29)), it must have the form

$$F(\mathbf{x}^{[1,d-1]}, \mathbf{x}^d, \mathbf{x}^{[d+1,D]}) = \langle \mathbf{p}^d, \mathbf{x}^d \rangle + \text{cst}(\mathbf{x}^{\text{d}}), \tag{6.40}$$

where $\text{cst}(\mathbf{x}^{\text{d}})$ is a term that does not depend on $\mathbf{x}^d$. Indeed, it can be shown (detailed in the appendix) that $\mathbf{p}^d = (\mathbf{p}_1^d, \dots, \mathbf{p}_n^d)$ where

$$\mathbf{p}_i^d = \sum_{\alpha=d}^D \left(\sum_{i_1 \dots i_{d-1} i i_{d+1} \dots i_\alpha \in \mathcal{C}} \mathbf{F}_{i_1 i_2 \dots i_\alpha} \bigotimes \left\{ \mathbf{x}_{i_1}^1, \dots, \mathbf{x}_{i_{d-1}}^{d-1}, \mathbf{x}_{i_{d+1}}^{d+1}, \dots, \mathbf{x}_{i_\alpha}^\alpha \right\} \right) \forall i \in \mathcal{V}. \tag{6.41}$$

While the expression of $\mathbf{p}_i^d$ looks complicated, its intuition is simple: for a given node i and a degree d, we search for all cliques satisfying two conditions: (a) their sizes are bigger than or equal to d, and (b) the node i is at the d^{th} position of these cliques; then for each clique, we multiply its potential tensor with all its nodes except node i, and sum all these products together.

Denote

$$\mathbf{s}^d = \sum_{c=1}^{d-1} \mathbf{A}^c \mathbf{x}^c + \sum_{c=d+1}^{D} \mathbf{A}^c \mathbf{x}^c. \tag{6.42}$$

Plugging (6.40) and (6.42) into (6.36) we get:

$$L_\rho(\mathbf{x}^1, \dots, \mathbf{x}^D, \mathbf{y}) = \frac{\rho}{2}\left\|\mathbf{A}^d \mathbf{x}^d\right\|_2^2 + \left(\mathbf{p}^d + \mathbf{A}^{d\top}\mathbf{y} + \rho \mathbf{A}^{d\top}\mathbf{s}^d\right)^\top \mathbf{x}^d + \mathrm{cst}(\mathbf{x}^{\mathrm{d}}). \tag{6.43}$$

Therefore, the $\mathbf{x}$ update (6.37) becomes minimizing the quadratic function (6.43) (with respect to $\mathbf{x}^d$) over $\mathcal{X}^d$. With suitable decompositions, this problem can have a much simpler form and can be efficiently solved. For example, if we choose the *cyclic* decomposition (6.33), then this step is reduced to finding the projection of a vector onto $\mathcal{X}^d$:

$$\mathbf{x}^{d^{(k+1)}} = \operatorname*{argmin}_{\mathbf{x}^d \in \mathcal{X}^d} \left\|\mathbf{x}^d - \mathbf{c}^{d^{(k)}}\right\|_2^2, \tag{6.44}$$

where $(\mathbf{c}_d)_{1\le d\le D}$ are defined as follows (*c.f.* appendix):

$$\mathbf{c}^{1^{(k)}} = \mathbf{x}^{2^{(k)}} - \frac{1}{\rho}\left(\mathbf{y}^{2^{(k)}} + \mathbf{p}^{1^{(k)}}\right), \tag{6.45}$$

$$\mathbf{c}^{d^{(k)}} = \frac{1}{2}\left(\mathbf{x}^{d-1^{(k+1)}} + \mathbf{x}^{d+1^{(k)}}\right) + \frac{1}{2\rho}\left(\mathbf{y}^{d^{(k)}} - \mathbf{y}^{d+1^{(k)}} - \mathbf{p}^{d^{(k)}}\right), \quad 2 \le d \le D-1, \tag{6.46}$$

$$\mathbf{c}^{D^{(k)}} = \mathbf{x}^{D-1^{(k+1)}} + \frac{1}{\rho}\left(\mathbf{y}^{D^{(k)}} + \mathbf{p}^{D^{(k)}}\right). \tag{6.47}$$

Here the multiplier $\mathbf{y}$ is the concatenation of $(D-1)$ vectors $(\mathbf{y}^d)_{2\le d\le D}$, corresponding to $(D-1)$ constraints in (6.33).

Similar results can be obtained for other specific decompositions such as *star* (6.34) and *symmetric* (6.35) as well. We refer to Appendix B.2.2 for more details. As we observed very similar performance among these decompositions, only *cyclic* was included for evaluation (Section 6.6).

The ADMM procedure are sketched in Algorithm 6.4.

ALGORITHM 6.4 ADMM with general decomposition (6.30) for solving (RLX).

1: Initialization: $k \leftarrow 0$, $\mathbf{y}^{(0)} \leftarrow \mathbf{0}$ and $\mathbf{x}^{d^{(0)}} \in \mathcal{X}^d$ for $d = 1, \dots, D$.
2: **for** $d = 1, 2, \dots, D$ **do**
3: Update $\mathbf{x}^{d^{(k+1)}}$ by solving (6.37) (*i.e.* minimizing (6.43) over $\mathcal{X}^d$).
4: **end for**
5: Update $\mathbf{y}^{(k+1)}$ using (6.38). Let $k \leftarrow k+1$ and go to Step 2.

In practice, we found that the penalty parameter ρ and the constraint sets $(\mathcal{X}^d)_{1\le d\le D}$ can greatly affect the convergence as well as the solution quality of ADMM. Let us address these together with other practical considerations.

Adaptive penalty We observed that small ρ leads to slower convergence but often better energy, and inversely for large ρ. To obtain a good trade-off, we use the same adaptive scheme presented in Chapter 5 (Section 5.3.5): initialize ρ at a small value ρ_0 and run for I_1 iterations, after that if no improvement of the residual $r^{(k)}$ is achieved

every I_2 iterations, then we increase ρ by a factor β. In addition, we stop increasing ρ after it reaches some value $\rho_{\max}$, so that the convergence properties presented in the next section still apply. In the experiments, we normalize all the potentials to $[-1, 1]$ and set $I_1 = 500, I_2 = 500, \beta = 1.2, \rho_0 = 0.001, \rho_{\max} = 100$.

Constraint sets A trivial choice of $(\mathcal{X}^d)_{1 \leq d \leq D}$ that satisfies (6.32) is $\mathcal{X}^d = \mathcal{X}\ \forall d$. Then, (6.44) becomes projections onto the simplex $\{\mathbf{x}_i \mid \mathbf{1}^\top \mathbf{x}_i = 1, \mathbf{x}_i \geq \mathbf{0}\}$ for each node i, which can be solved using *e.g.* the method introduced in [Condat, 2016]. However, we found that this choice often produces poor quality solutions, despite converging quickly. The reason is that constraining all $\mathbf{x}_i^d$ to belong to a simplex will make them reach consensus faster, but without being allowed to vary more freely, they tend to bypass good solutions. The idea is to use looser constraint sets, *e.g.* $\mathcal{X}^+ := \{\mathbf{x} \mid \mathbf{x} \geq \mathbf{0}\}$, for which (6.44) becomes simply $\mathbf{x}^{d^{(k+1)}} = \max(\mathbf{c}^{d^{(k)}}, 0)$. We found that leaving only one set as $\mathcal{X}$ yields the best accuracy. Therefore, in our implementation we set $\mathcal{X}^1 = \mathcal{X}$ and $\mathcal{X}^d = \mathcal{X}^+\ \forall d \geq 2$.

Parallelization Since there is no dependency among the nodes in the constraint sets, the projection (6.44) is clearly reduced to *independent* projections at each node. Moreover, at each iteration, the expensive computation (6.41) of $\mathbf{p}_i^d$ can also be performed in parallel for all nodes. Therefore, the proposed ADMM is highly parallelizable.

Caching Significant speed-ups can be achieved by avoiding re-computation of unchanged quantities. From (6.41) it is seen that $\mathbf{p}_i^d$ only depends on the decomposed variables at the neighbors of i. Thus, if these variables have not changed from the last iteration, then there is no need to recompute $\mathbf{p}_i^d$ in the current iteration. Similarly, the projection (6.44) for $\mathbf{x}_i^d$ can be omitted if $\mathbf{c}_i^d$ is unchanged (*c.f.* (6.45)–(6.47)).

6.5 CONVERGENCE ANALYSIS

In this section, we establish some convergence results for the presented methods. Since the proofs of these results are rather long, they are given in Appendix B.

Definition 1 (Stationary point)**.** Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuously differentiable function over a closed convex set $\mathcal{M}$. A point $\mathbf{u}^*$ is called a *stationary point* of the problem $\min_{\mathbf{u} \in \mathcal{M}} f(\mathbf{u})$ if and only if it satisfies

$$\nabla f(\mathbf{u}^*)^\top (\mathbf{u} - \mathbf{u}^*) \geq 0 \quad \forall \mathbf{u} \in \mathcal{M}. \tag{6.48}$$

Note that (6.48) is a necessary condition for a point $\mathbf{u}^*$ to be a local optimum. This is a basic result and a proof can be found in *e.g.* [Bertsekas, 1999, Chapter 2].

Proposition 6.2. *Let $\{\mathbf{x}^{(k)}\}$ be a sequence generated by BCD, PGD or FW (Algorithms 6.1, 6.2 or 6.3) with line-search (6.19). Then every limit point[3] of $\{\mathbf{x}^{(k)}\}$ is stationary.*

Proof. See Appendix B.1.3. □

[3] A vector $\mathbf{x}$ is a limit point of a sequence $\{\mathbf{x}^{(k)}\}$ if there exists a subsequence of $\{\mathbf{x}^{(k)}\}$ that converges to $\mathbf{x}$.

Next, we give a convergence result for ADMM.

Definition 2 (Karush-Kuhn-Tucker (KKT) conditions)**.** A point $(\mathbf{x}^{*1}, \mathbf{x}^{*2}, \dots, \mathbf{x}^{*D}, \mathbf{y}^*)$ is said to be a KKT point of Problem (6.30) if it satisfies the following KKT conditions:

$$\mathbf{x}^{*d} \in \mathcal{X}^d, \qquad d = 1, \dots, D, \tag{6.49}$$

$$\mathbf{A}^1\mathbf{x}^{*1} + \cdots + \mathbf{A}^D\mathbf{x}^{*D} = \mathbf{0}, \tag{6.50}$$

$$\mathbf{x}^{*d} \in \underset{\mathbf{x}^d \in \mathcal{X}^d}{\operatorname{argmin}} \left\{ F(\mathbf{x}^{*[1,d-1]}, \mathbf{x}^d, \mathbf{x}^{*[d+1,D]}) + \mathbf{y}^{*\top}\mathbf{A}^d\mathbf{x}^d \right\}. \tag{6.51}$$

Note that (6.50) is equivalent to $\mathbf{x}^{*1} = \mathbf{x}^{*2} = \cdots = \mathbf{x}^{*D}$ (because of (6.31)). Therefore, any KKT point of (6.30) must have the form $(\mathbf{x}^*, \dots, \mathbf{x}^*, \mathbf{y}^*)$ for some vector $\mathbf{x}^*$ and $\mathbf{y}^*$.

Proposition 6.3. *Let* $\{(\mathbf{x}^{1^{(k)}}, \dots, \mathbf{x}^{D^{(k)}}, \mathbf{y}^{(k)})\}$ *be a sequence generated by ADMM (Algorithm 6.4). Assume that the residual* $r^{(k)}$ (6.39) *converges to* 0*, then any limit point of this sequence is a KKT point of* (6.30)*.*

Proof. See Appendix B.1.4. □

We should note that this result is only partial, since we need the assumption that $r^{(k)}$ converges to 0. In practice, we found that this assumption always holds if ρ is large enough. Unlike gradient methods, convergence of ADMM for the kind of Problem (6.30) (which is at the same time multi-block, non-separable and highly nonconvex) is less known and is a current active research topic (*c.f.* Chapter 4). For example, global convergence of ADMM for nonconvex nonsmooth functions is established in [Wang et al., 2015b], but under numerous assumptions that are not applicable to our case, as presented in Section 4.2.3, Table 4.1.

So far for ADMM we have talked about solution to (6.30) only and not to (RLX). In fact, we have the following result.

Proposition 6.4. *If* $(\mathbf{x}^*, \mathbf{x}^*, \dots, \mathbf{x}^*, \mathbf{y}^*)$ *is a KKT point of* (6.30) *then* $\mathbf{x}^*$ *is a stationary point of* (RLX)*.*

Proof. See Appendix B.1.5. □

An interesting relation of the solutions returned by the methods is the following. We say a method A can improve further a method B if we use the returned solution by B as initialization for A and A will output a better solution.

Proposition 6.5. *At convergence:*

1. *BCD, PGD and FW cannot improve further each other.*
2. *BCD, PGD and FW cannot improve further ADMM. The inverse is not necessarily true.*

Proof. It is straightforward to see that the first point follows from the fact that solutions of BCD, PGD and FW are stationary, and the second point follows from Proposition 6.4. □

In practice, we observed that ADMM can often improve further the other methods.

6.6 EXPERIMENTS

In this section we evaluate our proposed nonconvex relaxation algorithms:

- Block Coordinate Descent (**BCD**.
- Projected Gradient Descent (**PGD**).
- Frank-Wolfe algorithm (**FW**).
- Nonconvex Alternating Direction Method of Multipliers (**ADMM**) with cyclic decomposition.

We compare them with the following state-of-the-art methods.

Pairwise methods:

- α-Expansion (α**-Exp**) [Boykov et al., 2001].
- Fast Primal-Dual (**FastPD**) [Komodakis et al., 2008].
- Convex QP Relaxation (**CQP**) [Ravikumar and Lafferty, 2006].
- Sequential Tree Reweighted Message Passing (**TRWS**) [Kolmogorov, 2006].

Higher-order methods:

- Tree Reweighted Belief Propagation (**TRBP**) [Wainwright et al., 2005].
- Alternating Direction Dual Decomposition (**ADDD**) [Martins et al., 2015].
- Bundle Dual Decomposition[4] (**BUNDLE**) [Kappes et al., 2012].
- Max-Product Linear Programming (**MPLP**) [Globerson and Jaakkola, 2008] and its extension (**MPLP-C**) [Sontag et al., 2012].
- Order reduction and α-expansion (α**-Fusion**) [Fix et al., 2011], which can be seen as an extension of α-expansion to higher-order.
- Sequential Reweighted Message Passing (**SRMP**) [Kolmogorov, 2015], which is a direct generalization of TRWS to higher-order.

The software for most methods are obtained via either the popular OpenGM library [Andres et al., 2012] or from the corresponding authors' websites, except for CQP [Ravikumar and Lafferty, 2006] we use our implementation as no code is publicly available (*c.f.* Appendix B.2.1 for implementation details).

[4]Subgradient dual decomposition [Komodakis et al., 2011] is excluded as we found that its performance was generally worse than bundle dual decomposition.

For BCD, PGD and FW, we run for 5 different initializations (solution of the unary potentials plus 4 other completely random) and pick the best one. For ADMM, we use a *single* homogeneous initial solution: $x_i(s) = \frac{1}{|\mathcal{S}_i|} \forall s \in \mathcal{S}_i$ (we find that ADMM is quite insensitive to initialization). For these methods, BCD is used as a final rounding step. We should note that BCD cannot improve further the solution according to Proposition 6.5.

TABLE 6.1 List of models used for evaluation.

Model	No.*	$\lvert\mathcal{V}\rvert$**	$S^\dagger$	$D^\ddagger$	Structure	Function
Inpainting	4	14400	4	2	grid-N4/N8	Potts
Matching	4	∼20	∼20	2	full/sparse	general
1st stereo	3	∼100000	16-60	2	grid-N4	TL/TS
Segmentation	10	1024	4	4	grid-N4	g-Potts
2nd stereo	4	∼25000	14	3	grid-N4	general

*,**,†,‡: number of instances, variables, labels, and MRF degree

The methods are evaluated on several real-world vision tasks: image inpainting, feature matching, image segmentation and stereo reconstruction. All methods are included whenever applicable, except when there are duplicates in pairwise settings where a pairwise algorithm and its higher-order generalization can be both applied, we include only the pairwise version as it is better optimized. A summary of the models are given in Table 6.1. Except for higher-order stereo, these models were previously considered in a recent benchmark for evaluating MRF optimization methods [Kappes et al., 2015], and their model files are publicly available[5]. For higher-order stereo, we replicate the model presented in [Woodford et al., 2009], where the disparity map is encouraged to be piecewise smooth using a second-order prior, and the labels are obtained from 14 pre-generated piecewise-planar proposals. We apply this model to 4 image pairs (*art, cones, teddy, venus*) from the Middlebury dataset [Scharstein and Szeliski, 2003] (at half resolution, due to the high inference time). We refer to [Kappes et al., 2015] and [Woodford et al., 2009] for further details.

The experiments were carried out on a 64-bit Linux machine with a 3.4GHz processor and 32GB of memory. A time limit of 1 hour was set for all methods. In Tables 6.2, 6.3 and 6.4, we report the runtime[6], the energy value of the final integer solution as well as the lower bound if available, averaged over all instances of a particular model. The detailed results are given in Appendix B.3.

In general, ADMM significantly outperforms BCD, PGD, FW and is the only nonconvex relaxation method that compares favorably with the other methods. In particular, it outperforms TRBP, ADDD, BUNDLE, MPLP, MPLP-C and CQP on all models (except MPLP-C on matching), and outperforms FastPD, α-Exp/α-Fusion and TRWS on small or medium sized models (*i.e.* other than stereo).

On image inpainting (Table 6.2), ADMM produces the lowest energies on all instances, while being relatively fast. Surprisingly TRWS performs poorly on these models, even worse than BCD, PGD and FW.

[5] http://hciweb2.iwr.uni-heidelberg.de/opengm/index.php?l0=benchmark

[6] For a fair comparison, we used the *single-thread* version of ADMM.

TABLE 6.2 Results on pairwise inpainting models.

	Inpainting N4 (2 instances)			Inpainting N8 (2 instances)		
algorithm	time (s)	value	bound	time (s)	value	bound
α-Exp	0.02	**454.35**	$-\infty$	0.78	465.02	$-\infty$
FastPD	0.03	454.75	294.89	0.15	465.02	136.28
TRBP	23.45	480.27	$-\infty$	64.00	495.80	$-\infty$
ADDD	15.87	483.41	443.71	35.78	605.14	450.95
MPLP	55.32	497.16	411.94	844.97	468.97	453.55
MPLP-C	1867.20	468.88	448.03	2272.39	479.54	454.35
BUNDLE	36.18	455.25	448.23	111.74	465.26	455.43
TRWS	1.37	490.48	448.09	16.23	500.09	453.96
CQP	1.92	1399.51	$-\infty$	11.62	1178.91	$-\infty$
BCD	0.11	485.88	$-\infty$	0.29	481.95	$-\infty$
FW	1.10	488.23	$-\infty$	5.94	489.82	$-\infty$
PGD	0.81	489.80	$-\infty$	5.19	489.82	$-\infty$
ADMM	9.84	**454.35**	$-\infty$	40.64	**464.76**	$-\infty$

TABLE 6.3 Results on pairwise matching and stereo models.

	Feature matching (4 instances)			Pairwise stereo (3 instances)		
algorithm	time (s)	value	bound	time (s)	value	bound
α-Exp	–*	–*	–*	14.75	1617196.00	$-\infty$
FastPD	–*	–*	–*	7.14	1614255.00	301059.33
TRBP	0.00	1.05×10^{11}	$-\infty$	2544.12	1664504.33	$-\infty$
ADDD	3.16	1.05×10^{11}	16.35	–**	–**	–**
MPLP	0.47	0.65×10^{11}	15.16	–**	–**	–**
MPLP-C	6.04	**21.22**	21.22	–**	–**	–**
BUNDLE	2.33	0.10×10^{11}	14.47	2039.47	1664707.67	1583742.13
TRWS	0.05	64.19	15.22	421.20	**1587961.67**	1584746.58
CQP	0.08	127.01	$-\infty$	3602.01	11408446.00	$-\infty$
BCD	0.00	84.86	$-\infty$	10.82	7022189.00	$-\infty$
FW	20.10	66.71	$-\infty$	1989.12	6162418.00	$-\infty$
PGD	13.21	58.52	$-\infty$	1509.49	5209092.33	$-\infty$
ADMM	0.31	75.12	$-\infty$	2377.66	1624106.00	$-\infty$

*Method not applicable

**Prohibitive execution time (time limit not working) or prohibitive memory consumption

The feature matching model (Table 6.3) is a typical example showing that the standard LP relaxation can be very loose. All methods solving its dual produce very poor results (despite reaching relatively good lower bounds). They are largely outperformed by TRWS and nonconvex relaxation methods (BCD, PGD, FW, ADMM). On this problem, MPLP-C reaches the global optimum for all instances.

On image segmentation (Table 6.4), SRMP performs exceptionally well, producing the global optimum for all instances while being very fast. ADMM is only slightly outperformed by SRMP in terms of energy value, while both clearly outperform the

TABLE 6.4 Results on higher-order models.

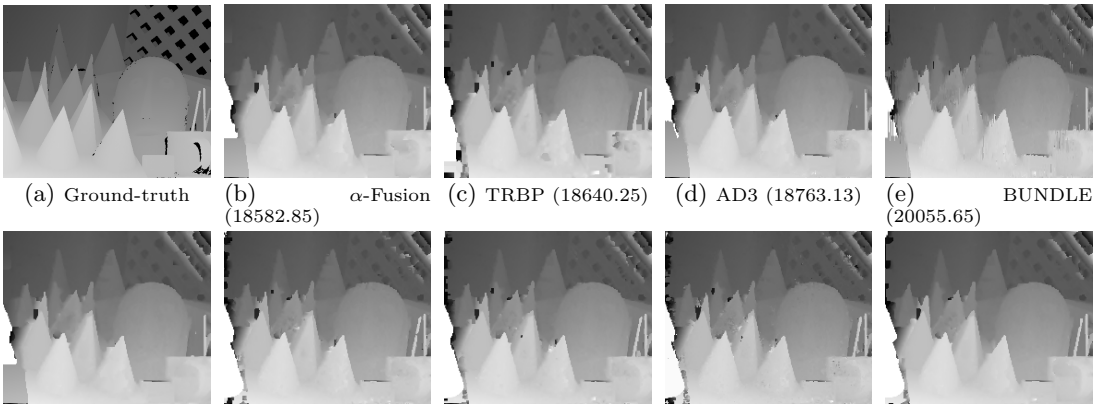
	Segmentation (10 instances)			Second-order stereo (4 instances)		
algorithm	time (s)	value	bound	time (s)	value	bound
α-Fusion	0.05	1587.13	$-\infty$	50.03	14035.91	$-\infty$
TRBP	18.20	1900.84	$-\infty$	3675.90	14087.40	$-\infty$
ADDD	6.36	3400.81	1400.33	4474.83	14226.93	13752.73
MPLP	9.68	4000.44	1400.30	–*	–*	–*
MPLP-C	3496.50	4000.41	1400.35	–*	–*	–*
BUNDLE	101.56	4007.73	1392.01	3813.84	15221.19	13321.96
SRMP	0.13	**1400.57**	1400.57	3603.41	**13914.82**	13900.87
BCD	0.14	12518.59	$-\infty$	59.59	14397.22	$-\infty$
FW	21.23	5805.17	$-\infty$	1749.19	14272.54	$-\infty$
PGD	51.04	5513.02	$-\infty$	3664.92	14543.65	$-\infty$
ADMM	97.37	1400.68	$-\infty$	3662.13	14068.53	$-\infty$

*Prohibitive execution time (time limit not working)

other methods.

On large scale models such as stereo (Tables 6.3 and 6.4), TRWS/SRMP perform best in terms of energy value, followed by move making algorithms (FastPD, α-Exp/α-Fusion) and ADMM. An example of estimated disparity maps is given in Figure 6.1.

An interesting observation is that CQP performs worse than nonconvex methods on all models (and worst overall), which means simply solving the QP relaxation in a straightforward manner is already better than adding a sophisticated convexification step, as done in [Ravikumar and Lafferty, 2006].



(a) Ground-truth (b) α-Fusion (18582.85) (c) TRBP (18640.25) (d) AD3 (18763.13) (e) BUNDLE (20055.65)

(f) SRMP (**18433.01**) (g) BCD (18926.70) (h) FW (18776.26) (i) PGD (19060.17) (j) ADMM (18590.87)

FIGURE 6.1 Resulted disparity maps and energy values using second-order MRFs for the *cones* scene of the Middlebury stereo dataset [Scharstein and Szeliski, 2003].

6.7 CONCLUSION

We have presented a tight nonconvex continuous relaxation for the problem of MAP inference and studied four different methods for solving it: block coordinate descent, projected gradient descent, Frank-Wolfe algorithm, and ADMM. Due to the high non-

convexity, it is very challenging to obtain good solutions to this relaxation, as shown by the performance of the first three methods. The latter, however, outperforms many existing methods and thus demonstrates that directly solving the nonconvex relaxation can lead to very accurate results. These methods are memory efficient, thanks to the small number of variables and constraints (as discussed in Section 6.3). On top of that, the proposed ADMM algorithm is also highly parallelizable (as discussed in Section 6.4.2), which is not the case for methods like TRWS or SRMP. Therefore, ADMM is also suitable for distributed or real-time applications on GPUs.

7

Deep Parameter Learning of Graph-Based Models

7.1 INTRODUCTION

We have seen that two major graph-based problems in computer vision — MAP inference and graph matching — can be both expressed as energy minimization:

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}\in\mathcal{X}} E(\mathbf{x}), \tag{7.1}$$

where $\mathcal{X}$ is a set representing structural constraints on $\mathbf{x}$. A more explicit formulation is the following:

$$\mathbf{x}^*(\mathbf{I};\boldsymbol{\theta}) = \operatorname*{argmin}_{\mathbf{x}\in\mathcal{X}} E(\mathbf{x},\mathbf{I};\boldsymbol{\theta}), \tag{7.2}$$

where $\mathbf{I}$ represents the input (*e.g.* an image or a set of extracted features) and $\boldsymbol{\theta}$ is the parameter of the model. More generally, problems of the form (7.2) are called *structured prediction*, which, by definition, aims to find an output $\mathbf{x}$ that best fits an input $\mathbf{I}$ where $\mathbf{x}$ obeys some structural constraints. In MAP inference or graph matching, $\boldsymbol{\theta}$ is specifically the so-called *potentials*.

In previous chapters, we have proposed methods for solving (7.2) — for MAP inference and graph matching — based on nonconvex ADMM. In all the experiments that we have seen, the parameters $\boldsymbol{\theta}$ were computed using hand-crafted formulas. In this chapter, we propose a method to learn them from training data, taking advantage of the proposed nonconvex ADMM inference framework.

Due to the popularity of graph-based models, there is a large body of literature on learning parameters of these models, especially for probabilistic graphical models (including MRFs and CRFs). We refer to [Koller and Friedman, 2009, Chapters 16–20] and [Nowozin et al., 2011] for in-depth reviews. The focus of this chapter will be on gradient-based learning which has become ubiquitous in machine learning and computer vision due to the wide spread adoption of deep neural networks over the last few years. Thanks to its flexibility, gradient-based learning allows combinations of different trainable models into single ones that can be trained in an end-to-end manner. An example of such combinations is between a convolutional neural network (CNN) and a CRF, as shown later in this chapter.

The core idea behind our method is to view ADMM as a sequence of simple differentiable operations, through which gradients can propagate (either forward or backward). Unrolling an optimization algorithm as a sequence of elementary differentiable operations and training using gradient back-propagation was proposed

in [Ross et al., 2011, Stoyanov et al., 2011] and [Domke, 2011] where the underlying inference algorithms are based on belief propagation. Later, [Domke, 2012] introduced similar ideas for end-to-end training of generic gradient-based inference algorithms such as gradient descent, heavy-ball or L-BFGS. These ideas have been applied to recent work such as deep energy models [Brakel et al., 2013], structured prediction energy networks [Belanger et al., 2017], and dense conditional random fields (CRFs) [Krähenbühl and Koltun, 2013]. In particular, the training framework proposed by [Krähenbühl and Koltun, 2013] for mean-field inference was later deployed in [Zheng et al., 2015] for training a combination of a CNN and a dense CRF. This combined model can be seen as a recurrent neural network (RNN), which allows automatic gradient computation when implementing using popular neural network libraries. It should be noted that the exact idea of viewing a graphical model as an RNN and using automatic differentiation was previously discussed in [Stoyanov et al., 2011] for belief propagation. Prior to that, the idea of unrolling message passing algorithms as simpler operations that can be performed within a CNN were introduced in [Tatikonda and Jordan, 2002].

In this chapter, we apply similar ideas for learning parameters of graph-based models, where nonconvex ADMM is used for inference. It turns out, however, that the nonconvex ADMM algorithms that we proposed in Chapters 5 and 6 lead to non-differentiable updates, which makes gradient-based learning invalid. To overcome this issue, we propose a modification to those algorithms by using a different penalty function that is carefully chosen so that ADMM updates become differentiable. This is also our major contribution in this work. The resulted learning framework is very general and allows training jointly graph-based models and other ones such as neural networks. Experiments on an semantic image segmentation dataset show that our method achieves superior results to mean-field inference based method [Krähenbühl and Koltun, 2013, Zheng et al., 2015].

In the next section, we give an introduction to empirical risk minimization by stochastic gradient descent, an extremely popular learning method that our framework is based on; we explain why it is necessary to compute derivatives involving the optimal solution of the prediction problem (7.2). In Section 7.3 we present implicit differentiation — a classical method for computing these derivatives — and discuss its fundamental limitations that motivate the idea of unrolled optimization, a better method for the task. Next, in Section 7.4 we show how to apply this idea to obtain a general theoretical framework for gradient computation when ADMM is used for inference. In Section 7.5, we argue that the nonconvex ADMM methods that we proposed in Chapters 5 and 6 are not applicable since they do not satisfy the differentiability assumption of the general framework. Therefore, in Section 7.6 we propose a solution to overcome this issue. Finally, in Section 7.7 we present an application of the proposed framework for semantic image segmentation, together with experimental results.

Notation

Before proceeding, for convenience let us recall some notations and introduce some new ones.

Recall that in MAP inference we consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a set of labels $\mathcal{S}$, while in graph matching we have two graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$.

In both cases the solution to our prediction problem can be represented by a binary assignment matrix $\mathbf{X}$ whose dimensions are respectively $|\mathcal{V}| \times |\mathcal{S}|$ or $|\mathcal{V}_1| \times |\mathcal{V}_2|$. In this chapter, we use the MAP inference notations for both cases (*i.e.* for graph matching: $\mathcal{V}_1 = \mathcal{V}, \mathcal{V}_2 = \mathcal{S}$ and $\mathbf{X}$ has additional constraints on its columns).

The main variable that we will be using is the assignment vector $\mathbf{x} := \text{vec}(\mathbf{X})$, a vectorized replica of $\mathbf{X}$. The vector $\mathbf{x}$ can be seen as a block vector where the i^{th} block corresponds to the i^{th} row in $\mathbf{X}$, which is also the assignment vector at the node i: $\mathbf{x}_i = (X_{is})_{s \in \mathcal{S}}$. The dimension of $\mathbf{x}$ is $n := |\mathcal{V}||\mathcal{S}|$.

We write $\mathbf{X} = \text{mat}(\mathbf{x})$ to indicate that $\mathbf{X}$ is a $|\mathcal{V}| \times |\mathcal{S}|$ reshaped version of $\mathbf{x}$. The mat operator is defined exclusively for vectors of dimension n. If $\mathbf{v} \in \mathbb{R}^n$ and $\mathbf{V} = \text{mat}(\mathbf{v})$ then an index a in $\mathbf{v}$ corresponds to an assignment $i \to s$ where $i \in \mathcal{V}, s \in \mathcal{S}$, *i.e.* $v_a = V_{is}$.

We use interchangeably $\mathbf{v}^\top \mathbf{w}$ and $\mathbf{v} \bullet \mathbf{w}$ to denote the dot product of two vectors $\mathbf{v}$ and $\mathbf{w}$. The element-wise product of two matrices $\mathbf{A}$ and $\mathbf{B}$ is denoted by $\mathbf{A} \odot \mathbf{B}$.

7.2 EMPIRICAL RISK MINIMIZATION AND STOCHASTIC GRADIENT DESCENT

Suppose that we are given a dataset $\{(\mathbf{I}^1, \hat{\mathbf{x}}^1), (\mathbf{I}^2, \hat{\mathbf{x}}^2), \dots, (\mathbf{I}^m, \hat{\mathbf{x}}^m)\}$ and a *loss function* $L(\mathbf{x}, \hat{\mathbf{x}})$ that measures the difference between the predicted output $\mathbf{x}$ and the true output $\hat{\mathbf{x}}$. We are interested in finding the value of $\boldsymbol{\theta}$ that minimizes the following quantity, called the **empirical risk** [Vapnik, 1992]:

$$R(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(\mathbf{x}^*(\mathbf{I}^i; \boldsymbol{\theta}), \hat{\mathbf{x}}^i). \tag{7.3}$$

To prevent overfitting, in practice, it is often preferred to minimize the sum of the empirical risk and a term $\lambda \Omega(\boldsymbol{\theta})$ called the *regularizer*, where $\lambda \in \mathbb{R}_+$ is the regularization coefficient that controls the relative importance between the two terms. The simplest and perhaps most commonly used regularizer is given by the squared ℓ_2-norm of the parameters:

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2} \|\boldsymbol{\theta}\|_2^2. \tag{7.4}$$

This is known as *weight decay* in the machine learning literature. To simplify the presentation, we omit the regularizer because it does not interfere directly with the derivation of the results.

A natural solution to empirical risk minimization is *gradient descent*, which consists of the following update steps:

$$\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} - \alpha^{(k)} \nabla_{\boldsymbol{\theta}} R(\boldsymbol{\theta}^{(k)}), \tag{7.5}$$

where k is the iteration counter and $\alpha^{(k)}$ is called the *step size*. In machine learning, $\alpha^{(k)}$ is usually called the *learning rate*. As a reminder, we have previously seen two variants of gradient descent in Section 6.4.1. In the sequel, we omit the iteration counter k for clarity purpose.

Obviously gradient descent is only possible if $R(\boldsymbol{\theta})$ is differentiable and the gradient $\nabla_{\boldsymbol{\theta}} R(\boldsymbol{\theta})$ can be evaluated at each step. In practice, $R(\boldsymbol{\theta})$ is often non-differentiable but only sub-differentiable. In that case we can still perform the above update step by replacing the gradient by a *subgradient* and the resulted method is called *subgradient method* (*c.f.* Section 4.1.1). For now, to simplify the presentation, let us **assume that** $R(\boldsymbol{\theta})$ **is differentiable** (and later, we will discuss considerations for non-differentiable case when it is necessary).

According to (7.3), $\nabla_{\boldsymbol{\theta}} R(\boldsymbol{\theta})$ can be computed as

$$\nabla_{\boldsymbol{\theta}} R(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}^*(\mathbf{I}^i; \boldsymbol{\theta}), \hat{\mathbf{x}}^i). \tag{7.6}$$

Clearly, this requires computing the gradient of the loss function for all training examples at each iteration. If the number of examples m is very large (*e.g.* billions), the time to take a single gradient step becomes prohibitively long, even if $\nabla_{\boldsymbol{\theta}} L(\mathbf{x}^*(\mathbf{I}^i; \boldsymbol{\theta}), \hat{\mathbf{x}}^i)$ can be computed efficiently. A solution to this is **stochastic gradient descent**. Instead of computing $\nabla_{\boldsymbol{\theta}} R(\boldsymbol{\theta})$ exactly, the idea is to compute at each iteration an estimate of it using a *minibatch* of training examples $\left\{(\mathbf{I}^1, \mathbf{x}^1), (\mathbf{I}^2, \hat{\mathbf{x}}^2), \dots, (\mathbf{I}^{m'}, \hat{\mathbf{x}}^{m'})\right\}$, where $m' \ll m$:

$$\mathbf{g} = \frac{1}{m'} \sum_{i=1}^{m'} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}^*(\mathbf{I}^i; \boldsymbol{\theta}), \hat{\mathbf{x}}^i), \tag{7.7}$$

and then perform the update (7.5) with $\mathbf{g}$ in place of $\nabla_{\boldsymbol{\theta}} R(\boldsymbol{\theta})$. A sketch of this algorithm is presented in Algorithm 7.1.

ALGORITHM 7.1 Stochastic gradient descent for empirical risk minimization.

1: Initialize the parameters $\boldsymbol{\theta}^{(0)}$ and the learning rate $\alpha^{(0)} > 0$.
2: **for** $k = 0, 1, 2, \dots$ **do**
3: Sample a minibatch of training examples $\left\{(\mathbf{I}^1, \mathbf{x}^1), (\mathbf{I}^2, \hat{\mathbf{x}}^2), \dots, (\mathbf{I}^{m'}, \mathbf{x}^{m'})\right\}$.
4: Choose a learning rate $\alpha^{(k)}$.
5: Compute an estimate of the gradient

$$\mathbf{g} \leftarrow \frac{1}{m'} \sum_{i=1}^{m'} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}^*(\mathbf{I}^i; \boldsymbol{\theta}), \hat{\mathbf{x}}^i),$$

6: Update $\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} - \alpha^{(k)} \mathbf{g}$.
7: **end for**

Stochastic gradient descent and its variants are extremely popular in machine learning, especially in deep learning. An overview of these algorithms can be found in [Goodfellow et al., 2016]. The convergence of stochastic gradient descent — in the sense that the expectation of the gradient norms cannot stay bounded away from zero — can hold under several assumptions, such as the objective function ($R(\boldsymbol{\theta})$ in our case) being continuously differentiable, its gradient being Lipschitz continuous, the learning rates following a diminishing scheme, and some others. We refer to [Bottou et al., 2018] for a theoretical and practical analysis of this algorithm for large-scale machine learning.

An important problem remains:

How do we compute the loss gradient $\nabla_{\boldsymbol{\theta}} L(\mathbf{x}^*(\mathbf{I};\boldsymbol{\theta}), \hat{\mathbf{x}})$?

Obviously this requires $\mathbf{x}^*(\mathbf{I};\boldsymbol{\theta})$ to be evaluated, *i.e.* one has to solve the energy minimization problem (7.2). In previous chapters we have proposed methods for this inference task based on nonconvex ADMM. In this chapter, we present a method for efficiently computing the loss gradient $\nabla_{\boldsymbol{\theta}} L(\mathbf{x}^*(\mathbf{I};\boldsymbol{\theta}), \hat{\mathbf{x}})$ when ADMM is the solver for (7.2), so that we can have a unified framework for inference and learning.

In the remainder of this chapter, for clarity let us omit the input $\mathbf{I}$ and the ground-truth prediction $\hat{\mathbf{x}}$ since they are not directly involved in the derivation. The symbol $\mathbf{I}$ is then reserved for the identity matrix whose dimension is understood from the context.

7.3 IMPLICIT DIFFERENTIATION AND UNROLLED OPTIMIZATION

First, we argue that the loss gradient $\nabla_{\boldsymbol{\theta}} L(\mathbf{x}^*(\boldsymbol{\theta}))$ can be easily computed in some cases, regardless of the method used for solving (7.2). For example, consider the case where the corresponding structured prediction task is an unconstrained optimization problem, *i.e.* $\mathcal{X} = \mathbb{R}^n$:

$$\mathbf{x}^*(\boldsymbol{\theta}) = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^n} E(\mathbf{x};\boldsymbol{\theta}). \tag{7.8}$$

A typical example is the Gaussian MRF, which encodes a Gaussian distribution of the form

$$p(\mathbf{x}) = \frac{1}{Z}\exp\left(-E(\mathbf{x};\boldsymbol{\theta})\right), \quad \text{with} \quad E(\mathbf{x};\boldsymbol{\theta}) = \frac{1}{2}\mathbf{x}^\top \mathbf{P}(\boldsymbol{\theta})\mathbf{x} + \mathbf{u}(\boldsymbol{\theta})^\top \mathbf{x}, \tag{7.9}$$

where $\mathbf{P}$ is a positive definite matrix for any parameter $\boldsymbol{\theta}$.

The optimality condition for (7.8) reads:

$$\frac{\partial E(\mathbf{x}^*(\boldsymbol{\theta});\boldsymbol{\theta})}{\partial \mathbf{x}} = \mathbf{0}. \tag{7.10}$$

Taking derivative with respect to $\boldsymbol{\theta}$ of both sides, and applying the chain rule, we obtain

$$\mathbf{0} = \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}} \frac{\partial E(\mathbf{x}^*(\boldsymbol{\theta});\boldsymbol{\theta})}{\partial \mathbf{x}} = \frac{\mathrm{d}\mathbf{x}^*(\boldsymbol{\theta})}{\mathrm{d}\boldsymbol{\theta}} \frac{\partial^2 E(\mathbf{x}^*(\boldsymbol{\theta});\boldsymbol{\theta})}{\partial \mathbf{x}^2} + \frac{\partial^2 E(\mathbf{x}^*(\boldsymbol{\theta});\boldsymbol{\theta})}{\partial \boldsymbol{\theta}\partial \mathbf{x}}. \tag{7.11}$$

Assume that the Hessian matrix of E (with respect to $\mathbf{x}$) is invertible — this is clearly the case for a Gaussian MRF —, the last equation yields

$$\frac{\mathrm{d}\mathbf{x}^*(\boldsymbol{\theta})}{\mathrm{d}\boldsymbol{\theta}} = -\frac{\partial^2 E(\mathbf{x}^*(\boldsymbol{\theta});\boldsymbol{\theta})}{\partial \boldsymbol{\theta}\partial \mathbf{x}} \left(\frac{\partial^2 E(\mathbf{x}^*(\boldsymbol{\theta});\boldsymbol{\theta})}{\partial \mathbf{x}^2}\right)^{-1}. \tag{7.12}$$

Therefore, we have

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}^*(\boldsymbol{\theta})) &= \frac{\mathrm{d}\mathbf{x}^*(\boldsymbol{\theta})}{\mathrm{d}\boldsymbol{\theta}} \frac{\partial L(\mathbf{x}^*(\boldsymbol{\theta}))}{\partial \mathbf{x}} \\ &= -\frac{\partial^2 E(\mathbf{x}^*(\boldsymbol{\theta});\boldsymbol{\theta})}{\partial \boldsymbol{\theta}\partial \mathbf{x}} \left(\frac{\partial^2 E(\mathbf{x}^*(\boldsymbol{\theta});\boldsymbol{\theta})}{\partial \mathbf{x}^2}\right)^{-1} \frac{\partial L(\mathbf{x}^*(\boldsymbol{\theta}))}{\partial \mathbf{x}}. \end{aligned} \tag{7.13}$$

Clearly, this result gives us a straightforward way to compute the loss gradient: first solve (7.8) to obtain $\mathbf{x}^*(\boldsymbol{\theta})$; then compute the above second-order derivatives and evaluate them at $\mathbf{x}^*(\boldsymbol{\theta})$; finally solve a linear system of the form $\mathbf{v} = \mathbf{A}^{-1}\mathbf{b}$ to compute $\nabla_{\boldsymbol{\theta}} L(\mathbf{x}^*(\boldsymbol{\theta}))$ according to (7.13).

The above approach was proposed in [Faugeras, 1993] for analyzing uncertainty in recovering 3D geometry, in [Tappen et al., 2007] for learning Gaussian CRFs, and was further discussed in [Domke, 2012]. For the more general case of constrained minimization (*i.e.* (7.2)), it is still possible to compute $\nabla_{\boldsymbol{\theta}} L(\mathbf{x}^*(\boldsymbol{\theta}))$ based on KKT conditions, under further assumptions (see *e.g.* [Gould et al., 2016]). Note that in the above theoretical derivation, it is assumed that the energy minimization (7.8) and the linear system (7.13) are solved exactly. Tolerances in one of these quantities can lead to very inaccurate gradient. Worse still, if the energy is nonconvex then it is generally impossible to solve (7.8) to global optimality.

To overcome the above limitations, [Domke, 2012] proposed a method called *back-optimization*, which consists of defining the loss in terms of the results of an incomplete optimization:

$$\mathbf{x}^*(\boldsymbol{\theta}) = \text{opt-alg}\, E(\mathbf{x}; \boldsymbol{\theta}), \tag{7.14}$$

where opt-alg denotes an operator that runs a given optimization algorithm for a specified number of iterations. For example, if the corresponding optimization algorithm is gradient descent consisting of updates

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \nabla_{\mathbf{x}} E(\mathbf{x}^{(k)}; \boldsymbol{\theta}), \tag{7.15}$$

then one can define $\mathbf{x}^*$ as the output of the algorithm after N iterations, *i.e.* $\mathbf{x}^* = \mathbf{x}^{(N)}$. Using the chain rule, one can compute $\frac{\partial L}{\partial \mathbf{x}^{(k)}}$ for all k using the following recursion:

$$\frac{\partial L}{\partial \mathbf{x}^{(k)}} = \frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{x}^{(k)}} \frac{\partial L}{\partial \mathbf{x}^{(k+1)}}. \tag{7.16}$$

The name *back-optimization* comes from this recursion where we start from $\frac{\partial L}{\partial \mathbf{x}^{(N)}}$ and arrive at $\frac{\partial L}{\partial \mathbf{x}^{(1)}}$. Finally $\nabla_{\boldsymbol{\theta}} L(\mathbf{x}^*(\boldsymbol{\theta}))$ can be computed using the chain rule again:

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \sum_{k=1}^{N} \frac{\partial \mathbf{x}^{(k)}}{\partial \boldsymbol{\theta}} \frac{\partial L}{\partial \mathbf{x}^{(k)}}. \tag{7.17}$$

The above idea can be applied to different optimization algorithms. In addition to gradient descent, [Domke, 2012] also studied second-order methods such as heavy-ball and L-BFGS, while [Krähenbühl and Koltun, 2013] for example applied it to mean-field inference in dense CRFs. In the next section, we present a general gradient computation framework for ADMM based on this idea. We unroll the algorithm with a fixed number of iterations to obtain a sequence of differentiable operations, and recursively compute derivatives using two different methods: forward-mode and reverse-mode differentiations (back-optimization in [Domke, 2012] is the latter). Actually, the general ideas of these two methods are the basis of *automatic differentiation* (also known as *autodiff*), a small but established field with applications in machine learning and other areas such as computational fluid dynamics, engineering design optimization, *etc.* (see *e.g.* [Baydin et al., 2018]). We compare and discuss the origin of these

two methods in Section 7.4.4.

7.4 GENERAL FRAMEWORK FOR ADMM GRADIENT COMPUTATION

7.4.1 *Unrolled ADMM and its computational graph*

First of all, we observe that there exist three functions f, g, h and three constant matrices/vector $\mathbf{A}, \mathbf{B}, \mathbf{c}$ such that the energy minimization problem (7.2) can be written under the following form, using a latent variable $\mathbf{z}$:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{z}} \quad & f(\mathbf{x};\boldsymbol{\theta}) + g(\mathbf{z};\boldsymbol{\theta}) + h(\mathbf{x},\mathbf{z};\boldsymbol{\theta}), \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} = \mathbf{c}. \end{aligned} \tag{7.18}$$

As an example, it is straightforward to see that (7.2) is equivalent to:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{z}} \quad & \delta_{\mathcal{X}}(\mathbf{x}) + \delta_{\mathcal{X}}(\mathbf{z}) + E\left(\frac{\mathbf{x}+\mathbf{z}}{2};\boldsymbol{\theta}\right), \\ \text{s.t.} \quad & \mathbf{x} = \mathbf{z}, \end{aligned} \tag{7.19}$$

where we recall that $\delta_{\mathcal{X}}(\cdot)$ denotes the indicator function of the set $\mathcal{X}$. Each choice of $(f, g, h, \mathbf{A}, \mathbf{B}, \mathbf{c})$ leads to a different formulation of (7.2) that we call a *decomposition*. In previous chapters we have seen different decompositions for MAP inference and graph matching that can leverage the effectiveness of nonconvex ADMM. In this section, for generality let us consider the decomposition (7.18).

Recall that ADMM solves (7.18) by iterating:

$$\mathbf{x}^{(k+1)} = \operatorname*{argmin}_{\mathbf{x}} L_\rho(\mathbf{x}, \mathbf{z}^{(k)}, \mathbf{y}^{(k)};\boldsymbol{\theta}), \tag{7.20}$$

$$\mathbf{z}^{(k+1)} = \operatorname*{argmin}_{\mathbf{z}} L_\rho(\mathbf{x}^{(k+1)}, \mathbf{z}, \mathbf{y}^{(k)};\boldsymbol{\theta}), \tag{7.21}$$

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + \rho\left(\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{z}^{(k+1)} - \mathbf{c}\right), \tag{7.22}$$

where L_ρ is the augmented Lagrangian:

$$L_\rho(\mathbf{x},\mathbf{z},\mathbf{y};\boldsymbol{\theta}) = f(\mathbf{x};\boldsymbol{\theta}) + g(\mathbf{z};\boldsymbol{\theta}) + h(\mathbf{x},\mathbf{z};\boldsymbol{\theta}) + \mathbf{y}^\top(\mathbf{A}\mathbf{x}+\mathbf{B}\mathbf{z}-\mathbf{c}) + \frac{\rho}{2}\|\mathbf{A}\mathbf{x}+\mathbf{B}\mathbf{z}-\mathbf{c}\|_2^2. \tag{7.23}$$

Denote

$$F_x(\mathbf{z},\mathbf{y};\boldsymbol{\theta}) = \operatorname*{argmin}_{\mathbf{x}} L_\rho(\mathbf{x},\mathbf{z},\mathbf{y};\boldsymbol{\theta}), \tag{7.24}$$

$$F_z(\mathbf{x},\mathbf{y};\boldsymbol{\theta}) = \operatorname*{argmin}_{\mathbf{z}} L_\rho(\mathbf{x},\mathbf{z},\mathbf{y};\boldsymbol{\theta}). \tag{7.25}$$

The above ADMM iterations become (7.26), (7.27), (7.28) as shown in Algorithm 7.2. These updates are represented by the computational graph in Figure 7.1.

Let us introduce an important assumption.

Assumption 7.1. *The functions F_x and F_z in (7.24) and (7.25) are differentiable and their gradients, namely $\nabla F_x := \left(\frac{\partial F_x}{\partial \mathbf{z}}, \frac{\partial F_x}{\partial \mathbf{y}}, \frac{\partial F_x}{\partial \boldsymbol{\theta}}\right)$ and $\nabla F_z := \left(\frac{\partial F_z}{\partial \mathbf{x}}, \frac{\partial F_z}{\partial \mathbf{y}}, \frac{\partial F_z}{\partial \boldsymbol{\theta}}\right)$, are*

ALGORITHM 7.2 Sketch of general ADMM for solving energy minimization.

1: Initialize $\mathbf{z}_0, \mathbf{y}_0$.
2: **for** $k = 0, 1, \dots, N-1$ **do**

$$\mathbf{x}^{(k+1)} \leftarrow F_x(\mathbf{z}^{(k)}, \mathbf{y}^{(k)}; \boldsymbol{\theta}), \tag{7.26}$$
$$\mathbf{z}^{(k+1)} \leftarrow F_z(\mathbf{x}^{(k+1)}, \mathbf{y}^{(k)}; \boldsymbol{\theta}), \tag{7.27}$$
$$\mathbf{y}^{(k+1)} \leftarrow \mathbf{y}^{(k)} + \rho(\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{z}^{(k+1)} - \mathbf{c}). \tag{7.28}$$

3: **end for**
4: Return the loss $L(\mathbf{x}^{(N)})$.

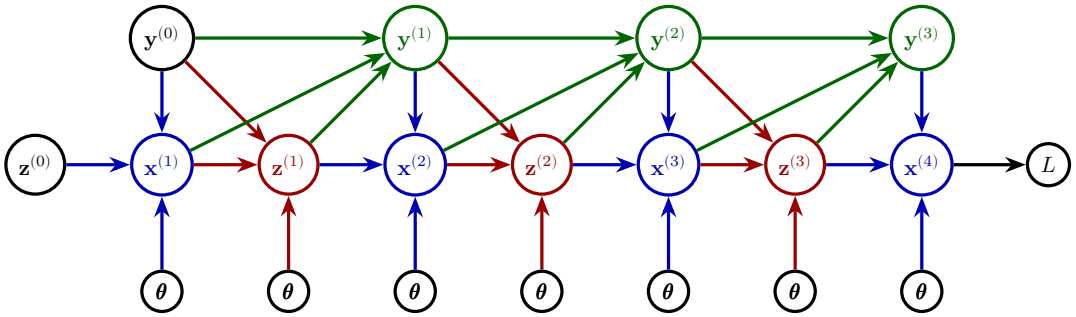


FIGURE 7.1 Computational graph illustrating ADMM for 4 iterations. Best viewed in color. The colors **blue**, **red** and **green** correspond respectively to the updates (7.26), (7.27) and (7.28) in Algorithm 7.2.

known.

Known gradients ∇F_x and ∇F_z can be achieved *e.g.* when F_x and F_z have analytic forms, *i.e.* when the minimization problems in the $\mathbf{x}$ and $\mathbf{z}$ update steps have closed-form solutions. If this is not the case then it is still possible to compute ∇F_x and ∇F_z based on optimality conditions of (7.24) and (7.25) (under further assumptions). We refer to [Gould et al., 2016] for a discussion on differentiating the argmin operator.

In the next sections we present two methods for computing $\nabla_{\boldsymbol{\theta}} L(\mathbf{x}^*(\boldsymbol{\theta}))$, under Assumption 7.1. We apply the same ideas of forward and backward modes in automatic differentiation [Baydin et al., 2018]. Since the ADMM iterates $\mathbf{x}^{(k)}, \mathbf{z}^{(k)}$ are defined directly by F_x, F_z:

$$\mathbf{x}^{(k+1)} = F_x(\mathbf{z}^{(k)}, \mathbf{y}^{(k)}, \boldsymbol{\theta}), \tag{7.29}$$
$$\mathbf{z}^{(k+1)} = F_z(\mathbf{x}^{(k+1)}, \mathbf{y}^{(k)}, \boldsymbol{\theta}), \tag{7.30}$$

we denote these gradients, evaluated at the previous iterates, using the derivatives of the current iterates. For example, we write $\frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{z}^{(k)}}$ to denote $\nabla_{\mathbf{z}} F_x(\mathbf{z}^{(k)}, \mathbf{y}^{(k)}, \boldsymbol{\theta})$.

7.4.2 *Forward-mode differentiation*

Consider a computational graph of nodes $(\mathbf{v}_i)_{i\in\mathcal{V}}$ in addition to an input node $\boldsymbol{\theta}$. Applying the derivative chain rule we have

$$\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}\boldsymbol{\theta}} = \sum_{j\in\mathrm{Pa}(i)} \frac{\mathrm{d}\mathbf{v}_j}{\mathrm{d}\boldsymbol{\theta}} \frac{\partial \mathbf{v}_i}{\partial \mathbf{v}_j}, \tag{7.31}$$

where $\mathrm{Pa}(i)$ denotes the set of indices of the parents of $\mathbf{v}_i$.

For any vector $\mathbf{v}$, denote

$$\dot{\mathbf{v}} := \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}\boldsymbol{\theta}}. \tag{7.32}$$

Applying (7.31) in turn for the nodes $\mathbf{x}^{(k+1)}, \mathbf{z}^{(k+1)}$ and $\mathbf{y}^{(k+1)}$ of the computational graph in Figure 7.1 we have:

$$\dot{\mathbf{x}}^{(k+1)} = \dot{\mathbf{z}}^{(k)} \frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{z}^{(k)}} + \dot{\mathbf{y}}^{(k)} \frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{y}^{(k)}} + \frac{\partial \mathbf{x}^{(k+1)}}{\partial \boldsymbol{\theta}}, \tag{7.33}$$

$$\dot{\mathbf{z}}^{(k+1)} = \dot{\mathbf{x}}^{(k+1)} \frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{x}^{(k+1)}} + \dot{\mathbf{y}}^{(k)} \frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{y}^{(k)}} + \frac{\partial \mathbf{z}^{(k+1)}}{\partial \boldsymbol{\theta}}, \tag{7.34}$$

$$\begin{aligned}\dot{\mathbf{y}}^{(k+1)} &= \dot{\mathbf{y}}^{(k)} \frac{\partial \mathbf{y}^{(k+1)}}{\partial \mathbf{y}^{(k)}} + \dot{\mathbf{x}}^{(k+1)} \frac{\partial \mathbf{y}^{(k+1)}}{\partial \mathbf{x}^{(k+1)}} + \dot{\mathbf{z}}^{(k+1)} \frac{\partial \mathbf{y}^{(k+1)}}{\partial \mathbf{z}^{(k+1)}}, \\ &= \dot{\mathbf{y}}^{(k)} + \rho \dot{\mathbf{x}}^{(k+1)} \mathbf{A}^\top + \rho \dot{\mathbf{z}}^{(k+1)} \mathbf{B}^\top,\end{aligned} \tag{7.35}$$

where in (7.33) and (7.34) we have implicitly used $\dot{\boldsymbol{\theta}} = \mathbf{I}$ (identity matrix), and in (7.35) we have used

$$\frac{\partial \mathbf{y}^{(k+1)}}{\partial \mathbf{x}^{(k+1)}} = \rho \mathbf{A}^\top, \quad \frac{\partial \mathbf{y}^{(k+1)}}{\partial \mathbf{z}^{(k+1)}} = \rho \mathbf{B}^\top, \quad \frac{\partial \mathbf{y}^{(k+1)}}{\partial \mathbf{y}^{(k)}} = \mathbf{I} \quad \forall k \geq 0, \tag{7.36}$$

which follow from (7.28). Let us recall that the partial derivatives in (7.33) and (7.34) are known, according to Assumption 7.1.

Clearly, the equations (7.33)–(7.35) say that $\left(\dot{\mathbf{x}}^{(k+1)}, \dot{\mathbf{z}}^{(k+1)}, \dot{\mathbf{y}}^{(k+1)}\right)$ can be computed from $\left(\dot{\mathbf{x}}^{(k)}, \dot{\mathbf{z}}^{(k)}, \dot{\mathbf{y}}^{(k)}\right)$, which implies that $\left(\dot{\mathbf{x}}^{(N)}, \dot{\mathbf{z}}^{(N)}, \dot{\mathbf{y}}^{(N)}\right)$ can be computed recursively from $\left(\dot{\mathbf{x}}^{(0)}, \dot{\mathbf{z}}^{(0)}, \dot{\mathbf{y}}^{(0)}\right)$.

Algorithm 7.3 shows how ADMM and its forward differentiation can be put together to compute the loss and its gradient.

7.4.3 *Reverse-mode differentiation*

For a general computational graph with output L, the derivative of L with respect to any node $\mathbf{v}_i$ satisfies the following identity, according to the chain rule:

$$\frac{\partial L}{\partial \mathbf{v}_i} = \sum_{j\in\mathrm{Ch}(i)} \frac{\partial \mathbf{v}_j}{\partial \mathbf{v}_i} \frac{\partial L}{\partial \mathbf{v}_j}, \tag{7.43}$$

where $\mathrm{Ch}(i)$ denotes the set of indices of the children of $\mathbf{v}_i$.

ALGORITHM 7.3 ADMM with forward-mode differentiation.

1: Initialize $\mathbf{z}_0, \mathbf{y}_0$ and set $\dot{\mathbf{z}}^{(0)} = \mathbf{0}, \dot{\mathbf{y}}^{(0)} = \mathbf{0}$.
2: **for** $k = 0, 1, \ldots, N-1$ **do**

$$\mathbf{x}^{(k+1)} \leftarrow F_x(\mathbf{z}^{(k)}, \mathbf{y}^{(k)}; \boldsymbol{\theta}), \tag{7.37}$$

$$\dot{\mathbf{x}}^{(k+1)} \leftarrow \dot{\mathbf{z}}^{(k)} \frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{z}^{(k)}} + \dot{\mathbf{y}}^{(k)} \frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{y}^{(k)}} + \frac{\partial \mathbf{x}^{(k+1)}}{\partial \boldsymbol{\theta}}, \tag{7.38}$$

$$\mathbf{z}^{(k+1)} \leftarrow F_z(\mathbf{x}^{(k+1)}, \mathbf{y}^{(k)}; \boldsymbol{\theta}), \tag{7.39}$$

$$\dot{\mathbf{z}}^{(k+1)} \leftarrow \dot{\mathbf{x}}^{(k+1)} \frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{x}^{(k+1)}} + \dot{\mathbf{y}}^{(k)} \frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{y}^{(k)}} + \frac{\partial \mathbf{z}^{(k+1)}}{\partial \boldsymbol{\theta}}, \tag{7.40}$$

$$\mathbf{y}^{(k+1)} \leftarrow \mathbf{y}^{(k)} + \rho(\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{z}^{(k+1)} - \mathbf{c}), \tag{7.41}$$

$$\dot{\mathbf{y}}^{(k+1)} \leftarrow \dot{\mathbf{y}}^{(k)} + \rho \dot{\mathbf{x}}^{(k+1)} \mathbf{A}^\top + \rho \dot{\mathbf{z}}^{(k+1)} \mathbf{B}^\top. \tag{7.42}$$

3: **end for**
4: Return the loss $L := L(\mathbf{x}^{(N)})$ and its gradient $\nabla_{\boldsymbol{\theta}} L := \dot{\mathbf{x}}^{(N)}$.

For convenience, denote

$$\bar{\mathbf{v}} := \frac{\partial L}{\partial \mathbf{v}}, \tag{7.44}$$

which is also called the *adjoint* vector of $\mathbf{v}$ [Griewank, 2010]. Clearly our goal is to compute $\bar{\boldsymbol{\theta}}$.

Applying (7.43) for the node $\boldsymbol{\theta}$ (*c.f.* the computational graph in Figure 7.1):

$$\bar{\boldsymbol{\theta}} = \frac{\partial \mathbf{x}^{(N)}}{\partial \boldsymbol{\theta}} \bar{\mathbf{x}}^{(N)} + \sum_{k=1}^{N-1} \left(\frac{\partial \mathbf{z}^{(k)}}{\partial \boldsymbol{\theta}} \bar{\mathbf{z}}^{(k)} + \frac{\partial \mathbf{x}^{(k)}}{\partial \boldsymbol{\theta}} \bar{\mathbf{x}}^{(k)} \right). \tag{7.45}$$

Applying (7.43) in turn for the nodes $\bar{\mathbf{x}}^{(k)}$, $\bar{\mathbf{y}}^{(k)}$ and $\bar{\mathbf{z}}^{(k)}$ we have:

$$\bar{\mathbf{x}}^{(k)} = \frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{x}^{(k)}} \bar{\mathbf{z}}^{(k)} + \frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{x}^{(k)}} \bar{\mathbf{y}}^{(k)}, \tag{7.46}$$

$$\bar{\mathbf{y}}^{(k)} = \frac{\partial \mathbf{y}^{(k+1)}}{\partial \mathbf{y}^{(k)}} \bar{\mathbf{y}}^{(k+1)} + \frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{y}^{(k)}} \bar{\mathbf{x}}^{(k+1)} + \frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{y}^{(k)}} \bar{\mathbf{z}}^{(k+1)}, \tag{7.47}$$

$$\bar{\mathbf{z}}^{(k)} = \frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{z}^{(k)}} \bar{\mathbf{x}}^{(k+1)} + \frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{z}^{(k)}} \bar{\mathbf{y}}^{(k)}. \tag{7.48}$$

Plugging (7.36) into the above equations we get:

$$\bar{\mathbf{x}}^{(k)} = \frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{x}^{(k)}} \bar{\mathbf{z}}^{(k)} + \rho \mathbf{A}^\top \bar{\mathbf{y}}^{(k)}, \tag{7.49}$$

$$\bar{\mathbf{y}}^{(k)} = \bar{\mathbf{y}}^{(k+1)} + \frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{y}^{(k)}} \bar{\mathbf{x}}^{(k+1)} + \frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{y}^{(k)}} \bar{\mathbf{z}}^{(k+1)}, \tag{7.50}$$

$$\bar{\mathbf{z}}^{(k)} = \frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{z}^{(k)}} \bar{\mathbf{x}}^{(k+1)} + \rho \mathbf{B}^\top \bar{\mathbf{y}}^{(k)}. \tag{7.51}$$

Similarly to the forward mode, here we have three recursive equations but in the

reverse direction. When going backward, the order of execution of these equations is important since the derivative at a node can only be evaluated given the derivatives at all of its children.

Algorithm 7.4 shows how ADMM and its reverse differentiation can be put together to compute the loss and its gradient.

ALGORITHM 7.4 ADMM with reverse-mode differentiation.

Forward pass:

1: Run Algorithm 7.2 to compute the loss $L(\mathbf{x}^{(N)})$. Store $\nabla F_x(\mathbf{z}^{(k)}, \mathbf{y}^{(k)}, \boldsymbol{\theta})$ and $\nabla F_z(\mathbf{x}^{(k+1)}, \mathbf{y}^{(k)}, \boldsymbol{\theta})$ in the memory for all k.

Reverse pass:

1: Initialization:

$$\bar{\mathbf{x}}^{(N)} \leftarrow \frac{\partial L}{\partial \mathbf{x}^{(N)}}, \tag{7.52}$$

$$\bar{\mathbf{y}}^{(N-1)} \leftarrow \frac{\partial \mathbf{x}^{(N)}}{\partial \mathbf{y}^{(N-1)}} \bar{\mathbf{x}}^{(N)}, \tag{7.53}$$

$$\bar{\boldsymbol{\theta}} \leftarrow \frac{\partial \mathbf{x}^{(N)}}{\partial \boldsymbol{\theta}} \bar{\mathbf{x}}^{(N)}. \tag{7.54}$$

2: **for** $k = (N-1), (N-2), \dots, 1$ **do**

$$\bar{\mathbf{z}}^{(k)} \leftarrow \frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{z}^{(k)}} \bar{\mathbf{x}}^{(k+1)} + \rho \mathbf{B}^\top \bar{\mathbf{y}}^{(k)}, \tag{7.55}$$

$$\bar{\mathbf{x}}^{(k)} \leftarrow \frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{x}^{(k)}} \bar{\mathbf{z}}^{(k)} + \rho \mathbf{A}^\top \bar{\mathbf{y}}^{(k)}, \tag{7.56}$$

$$\bar{\mathbf{y}}^{(k-1)} \leftarrow \bar{\mathbf{y}}^{(k)} + \frac{\partial \mathbf{x}^{(k)}}{\partial \mathbf{y}^{(k-1)}} \bar{\mathbf{x}}^{(k)} + \frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{y}^{(k-1)}} \bar{\mathbf{z}}^{(k)}, \tag{7.57}$$

$$\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \frac{\partial \mathbf{z}^{(k)}}{\partial \boldsymbol{\theta}} \bar{\mathbf{z}}^{(k)} + \frac{\partial \mathbf{x}^{(k)}}{\partial \boldsymbol{\theta}} \bar{\mathbf{x}}^{(k)} \tag{7.58}$$

3: **end for**

4: Return the loss $L := L(\mathbf{x}^{(N)})$ and its gradient $\nabla_{\boldsymbol{\theta}} L := \bar{\boldsymbol{\theta}}$.

7.4.4 *Forward mode or reverse mode?*

First, we observe that both methods perform the same operations needed to compute the output L (*i.e.* no derivative is involved) in the forward direction. In addition, when traversing the computational graph, both have to compute the Jacobian matrix $\frac{\partial \mathbf{v}_j}{\partial \mathbf{v}_i}$ for every edge $i \to j$. The difference is that the forward mode computes in addition $\frac{\partial \mathbf{v}_i}{\partial \boldsymbol{\theta}}$, while the reverse mode compute $\frac{\partial L}{\partial \mathbf{v}_i}$, for (almost) every node i. Suppose that $\boldsymbol{\theta} \in \mathbb{R}^p$ and $\mathbf{v}_i \in \mathbb{R}^{n_i}$ $\forall i$. For each edge $i \to j$, the forward mode performs a multiplication between two Jacobian matrices of dimensions $\mathbb{R}^{p \times n_i}$ and $\mathbb{R}^{n_i \times n_j}$ (according to (7.31)), while the reverse mode performs only a multiplication of an $\mathbb{R}^{n_i \times n_j}$ Jacobian matrix and an $\mathbb{R}^{n_j}$ vector (according to (7.43)). Therefore, it is clear that the reverse mode is more efficient in terms of computation operations. In contrast, the forward mode is more efficient in terms of memory consumption, because in the reverse mode, the

Jacobian matrices $\frac{\partial \mathbf{v}_j}{\partial \mathbf{v}_i}$ have to be computed during the forward pass and to be stored in the memory, awaiting for the reverse pass (*c.f.* Algorithm 7.4).

The above observations also hold for a general computational graph. In particular, if the dimensions of the input and output of a graph are respectively n_{in} and n_{out}, then the reverse mode is much more efficient if $n_{\text{in}} \gg n_{\text{out}}$, and vice-versa. This is a well-known result in automatic differentiation (see *e.g.* [Griewank, 2010, Baydin et al., 2018] for in-depth analysis and discussion). The training task in machine learning often involves the gradient of a scalar-valued objective with respect to a large number of parameters, which is the reason why reverse-mode differentiation is very popular in this field (although under different names until only recently). Perhaps its best known special case in machine learning is the *back-propagation* algorithm [LeCun et al., 1989] for training neural networks. Today it is widely known that reverse-mode differentiation was first introduced in [Linnainmaa, 1970].

7.5 ADMM FOR GRAPH-BASED MODELS: CURSE OF DIFFERENTIABILITY

In the previous section we have presented a general theoretical framework for structured prediction and learning with ADMM, in which we made a fundamental assumption on the differentiability of the $\mathbf{x}$ and $\mathbf{z}$ updates (*c.f.* Assumption 7.1). Unfortunately this assumption does not always hold in general. In this section, we will make this clear by trying applying the above framework to the methods that we have proposed in Chapters 5 and 6 for solving graph matching and MAP inference. Let us consider pairwise models for simplicity in the presentation. All results, however, can be trivially extended to higher-order models.

First, let us give a brief reminder from the previous chapters on how we solved the inference problem in question. Instead of solving the original discrete problem, we consider its *continuous relaxation*, which can be formulated as follows for (pairwise) MAP inference or graph matching:

$$\begin{aligned} \min \quad & E(\mathbf{x}) = \mathbf{u}^\top \mathbf{x} + \frac{1}{2}\mathbf{x}^\top \mathbf{P}\mathbf{x}, \\ \text{s.t.} \quad & \mathbf{x} \in \mathcal{X}, \end{aligned} \tag{7.59}$$

where $\mathbf{u}$ and $\mathbf{P}$ represent the unary and pairwise potentials of the model ($\mathbf{P}$ has zero diagonal), and $\mathcal{X}$ is the corresponding (relaxed) constraint set. Note that $\mathbf{P}$ and $\mathbf{u}$ are functions of the parameter $\boldsymbol{\theta}$, which we do not express explicitly for simplicity. The constraint set is different between MAP inference and graph matching:

$$\mathcal{X}_{\text{MAP}} = \left\{ \mathbf{x} \;\middle|\; \mathbf{1}^\top \mathbf{x}_i = 1, \mathbf{x}_i \geq \mathbf{0} \;\forall i \in \mathcal{V} \right\}, \tag{7.60}$$

$$\mathcal{X}_{\text{GM}} = \left\{ \mathbf{x} \geq \mathbf{0} \;\middle|\; \text{mat}(\mathbf{x}) \text{ obeys matching constraints} \right\}. \tag{7.61}$$

Here "matching constraints" can be any of one-to-one, one-to-many, or many-to-many constraints, *etc.*

Recall that in our proposed nonconvex ADMM inference method, we reformulate

the above problem as:

$$\begin{aligned} \min \quad & \mathbf{u}^\top \mathbf{x} + \frac{1}{2}\mathbf{x}^\top \mathbf{P}\mathbf{z}, \\ \text{s.t.} \quad & \mathbf{x} = \mathbf{z}, \\ & \mathbf{x} \in \mathcal{X}_1, \mathbf{z} \in \mathcal{X}_2, \end{aligned} \tag{7.62}$$

where $\mathcal{X}_1$ and $\mathcal{X}_2$ are closed convex sets satisfying $\mathcal{X}_1 \cap \mathcal{X}_2 = \mathcal{X}$.

The augmented Lagrangian is given by

$$L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{y}) = \mathbf{u}^\top \mathbf{x} + \frac{1}{2}\mathbf{x}^\top \mathbf{P}\mathbf{z} + \mathbf{y}^\top(\mathbf{x} - \mathbf{z}) + \frac{\rho}{2}\|\mathbf{x} - \mathbf{z}\|_2^2, \tag{7.63}$$

and the corresponding ADMM updates are:

$$\mathbf{x}^{(k+1)} = \underset{\mathbf{x}\in\mathcal{X}_1}{\operatorname{argmin}} \left\{ \left(\mathbf{u} + \frac{1}{2}\mathbf{P}\mathbf{z}^{(k)} + \mathbf{y}^{(k)} \right) \cdot \mathbf{x} + \frac{\rho}{2}\|\mathbf{x} - \mathbf{z}^{(k)}\|_2^2 \right\}, \tag{7.64}$$

$$\mathbf{z}^{(k+1)} = \underset{\mathbf{z}\in\mathcal{X}_2}{\operatorname{argmin}} \left\{ \left(\frac{1}{2}\mathbf{P}^\top \mathbf{x}^{(k+1)} - \mathbf{y}^{(k)} \right) \cdot \mathbf{z} + \frac{\rho}{2}\|\mathbf{z} - \mathbf{x}^{(k+1)}\|_2^2 \right\}, \tag{7.65}$$

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + \rho\left(\mathbf{x}^{(k+1)} - \mathbf{z}^{(k+1)}\right). \tag{7.66}$$

Our focus is on (7.64) and (7.65). It is easily seen that these steps can be reduced to the following projections:

$$\mathbf{x}^{(k+1)} = \underset{\mathbf{x}\in\mathcal{X}_1}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{v}^{(k+1)}\|_2^2, \tag{7.67}$$

$$\mathbf{z}^{(k+1)} = \underset{\mathbf{z}\in\mathcal{X}_2}{\operatorname{argmin}} \|\mathbf{z} - \mathbf{w}^{(k+1)}\|_2^2, \tag{7.68}$$

where

$$\mathbf{v}^{(k+1)} = \mathbf{z}^{(k)} - \frac{1}{\rho}\left(\mathbf{y}^{(k)} + \mathbf{u} + \frac{1}{2}\mathbf{P}\mathbf{z}^{(k)} \right), \tag{7.69}$$

$$\mathbf{w}^{(k+1)} = \mathbf{x}^{(k+1)} - \frac{1}{\rho}\left(-\mathbf{y}^{(k)} + \frac{1}{2}\mathbf{P}\mathbf{x}^{(k+1)} \right). \tag{7.70}$$

We have seen in Chapters 5 and 6 that the projections (7.67) and (7.68) can be solved using Lemma 5.1 (page 42). From the results obtained by this lemma, we observe that $\mathbf{x}^{(k+1)}$ and $\mathbf{z}^{(k+1)}$ are non-differentiable, despite having closed form solutions. Indeed, we give a proof of this result in Appendix C.1.1 (and a graphical illustration is presented later in Figure 7.2a). As a result, Assumption 7.1 is violated and therefore, we cannot apply the presented gradient computation framework.

In the next section, we propose a solution to overcome this issue.

7.6 BREGMAN ADMM: TOWARDS DIFFERENTIABLE UPDATES

We have seen briefly in Chapter 4 (Section 4.3.3) a generalization of ADMM called *Bregman ADMM*, which consists of replacing the ℓ_2-norm penalty term in the augmented Lagrangian by a more general distance function. The ℓ_2-norm penalty, as shown in the previous section, leads to projections onto convex sets, which make the $\mathbf{x}$

and $\mathbf{z}$ updates non-differentiable. A natural idea arises: if we are not restricted to the type of penalty functions, then possibly we can find one such that the ADMM updates become differentiable. It turns out that this idea is indeed valid.

7.6.1 Introduction to Bregman ADMM

Definition 3 ([Bregman, 1967, Censor and Zenios, 1997])**.** The *Bregman divergence* induced by a *continuously differentiable* and *strictly convex* function ϕ is defined by

$$D_\phi(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) - \phi(\mathbf{y}) - \langle \nabla\phi(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle . \tag{7.71}$$

Since ϕ is strictly convex, it is clear that $D_\phi(\mathbf{x}, \mathbf{y}) \geq 0 \ \forall \mathbf{x}, \mathbf{y}$ and equality occurs if and only if $\mathbf{x} = \mathbf{y}$. Examples of Bregman divergence generated from some convex functions are given in Table 7.1.

TABLE 7.1 Bregman divergence generated from some convex functions.

Domain	$\phi(\mathbf{x})$	$D_\phi(\mathbf{x}, \mathbf{y})$	Name
$\mathbb{R}^n$	$\|\mathbf{x}\|_2^2$	$\|\mathbf{x} - \mathbf{y}\|_2^2$	Euclidean distance
$\mathbb{R}^n$	$\mathbf{x}^\top \mathbf{M} \mathbf{x}$	$(\mathbf{x} - \mathbf{y})^\top \mathbf{M} (\mathbf{x} - \mathbf{y})$	Mahalanobis distance
simplex	$\sum_{i=1}^n x_i \log x_i$	$\sum_{i=1}^n x_i \log \left(\frac{x_i}{y_i}\right)$	Kullback-Leibler divergence
$\mathbb{R}_+^n$	$\sum_{i=1}^n x_i \log x_i$	$\sum_{i=1}^n \left(x_i \log \left(\frac{x_i}{y_i}\right) - x_i + y_i\right)$	Generalized KL-divergence

Naturally one may think about defining a new augmented Lagrangian based on Bregman divergence, *i.e.*

$$L_\rho^\phi(\mathbf{x}, \mathbf{z}, \mathbf{y}) = f(\mathbf{x}) + g(\mathbf{z}) + h(\mathbf{x}, \mathbf{z}) + \mathbf{y}^\top (\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}) + \rho D_\phi(\mathbf{c} - \mathbf{A}\mathbf{x}, \mathbf{B}\mathbf{z}), \tag{7.72}$$

and then perform the same updates as in regular ADMM. However, since the Bregman divergence is not necessarily convex with respect to the second argument, the $\mathbf{z}$-update might not be solved exactly to global optimum. A simple solution is to switch the order of the variables alternatingly: the $\mathbf{x}$-update uses $D_\phi(\mathbf{B}\mathbf{z}, \mathbf{c} - \mathbf{A}\mathbf{x})$ while the $\mathbf{z}$-update uses $D_\phi(\mathbf{c} - \mathbf{A}\mathbf{x}, \mathbf{B}\mathbf{z})$. The resulted algorithm is known as *Bregman ADMM* [Wang and Banerjee, 2014].

7.6.2 Differentiable Bregman ADMM for graph-based models

Now applying to (7.62), the corresponding Bregman ADMM updates read:

$$\mathbf{x}^{(k+1)} = \operatorname*{argmin}_{\mathbf{x} \in \mathcal{X}_1} \left\{ \mathbf{u}^\top \mathbf{x} + \frac{1}{2} \mathbf{x} \cdot \mathbf{P}\mathbf{z}^{(k)} + \mathbf{y}^{(k)} \cdot (\mathbf{x} - \mathbf{z}^{(k)}) + \rho D_\phi(\mathbf{x}, \mathbf{z}^{(k)}) \right\}, \tag{7.73}$$

$$\mathbf{z}^{(k+1)} = \operatorname*{argmin}_{\mathbf{z} \in \mathcal{X}_2} \left\{ \frac{1}{2} \mathbf{x}^{(k+1)} \cdot \mathbf{P}\mathbf{z} + \mathbf{y}^{(k)} \cdot (\mathbf{x}^{(k+1)} - \mathbf{z}) + \rho D_\phi(\mathbf{z}, \mathbf{x}^{(k+1)}) \right\}, \tag{7.74}$$

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + \rho(\mathbf{x}^{(k+1)} - \mathbf{z}^{(k+1)}), \tag{7.75}$$

which can be simplified as:

$$\mathbf{x}^{(k+1)} = \underset{\mathbf{x}\in\mathcal{X}_1}{\operatorname{argmin}} \left\{ \left(\mathbf{u} + \frac{1}{2}\mathbf{P}\mathbf{z}^{(k)} + \mathbf{y}^{(k)} \right) \cdot \mathbf{x} + \rho D_\phi(\mathbf{x}, \mathbf{z}^{(k)}) \right\}, \tag{7.76}$$

$$\mathbf{z}^{(k+1)} = \underset{\mathbf{z}\in\mathcal{X}_2}{\operatorname{argmin}} \left\{ \left(\frac{1}{2}\mathbf{P}^\top \mathbf{x}^{(k+1)} - \mathbf{y}^{(k)} \right) \cdot \mathbf{z} + \rho D_\phi(\mathbf{z}, \mathbf{x}^{(k+1)}) \right\}, \tag{7.77}$$

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + \rho \left(\mathbf{x}^{(k+1)} - \mathbf{z}^{(k+1)} \right). \tag{7.78}$$

As one may observe, the only difference compared to regular ADMM updates (7.64)–(7.66) is that here we have used the Bregman divergence penalty terms $D_\phi(\mathbf{x}, \mathbf{z}^{(k)})$ and $D_\phi(\mathbf{z}, \mathbf{x}^{(k+1)})$ instead of the ℓ_2-norm.

Now the most important question is: *How to choose the function ϕ such that the* argmin *operators in (7.76) and (7.77) are differentiable?* Below we give such an example of ϕ. The analysis of different Bregman divergences would be an interesting research direction that we leave for future work.

Consider the function $\phi : \mathbb{R}^n_+ \to \mathbb{R}$ defined by

$$\phi(\mathbf{x}) = \sum_{i=1}^{n} x_i \log x_i. \tag{7.79}$$

With simple calculations we obtain the Bregman divergence induced by ϕ:

$$D_\phi(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \left(x_i \log \left(\frac{x_i}{y_i} \right) - x_i + y_i \right). \tag{7.80}$$

The functions (7.79) and (7.80) are respectively known as the *negative entropy* and the (generalized) *Kullback-Leibler divergence* [Kullback and Leibler, 1951] (*c.f.* Table 7.1). The latter is usually denoted D_{KL}, so we adapt this notation in the sequel.

With the KL divergence, the updates (7.76) and (7.77) are indeed differentiable for the choices that we made on the decomposed sets $\mathcal{X}_1$ and $\mathcal{X}_2$ in Chapter 5 (for graph matching) and Chapter 6 (for MAP inference). This is clear thanks to the following lemma.

Lemma 7.1. *Let $\boldsymbol{\alpha} \in \mathbb{R}^p_+, \boldsymbol{\beta} \in \mathbb{R}^p$ be constant vectors. The optimal solution $\mathbf{w}^*$ of*

$$\min_{\mathbf{w}\in\mathcal{W}} \left\{ D_{\mathrm{KL}}(\mathbf{w}, \boldsymbol{\alpha}) - \boldsymbol{\beta}^\top \mathbf{w} \right\} \tag{7.81}$$

is given by the following.

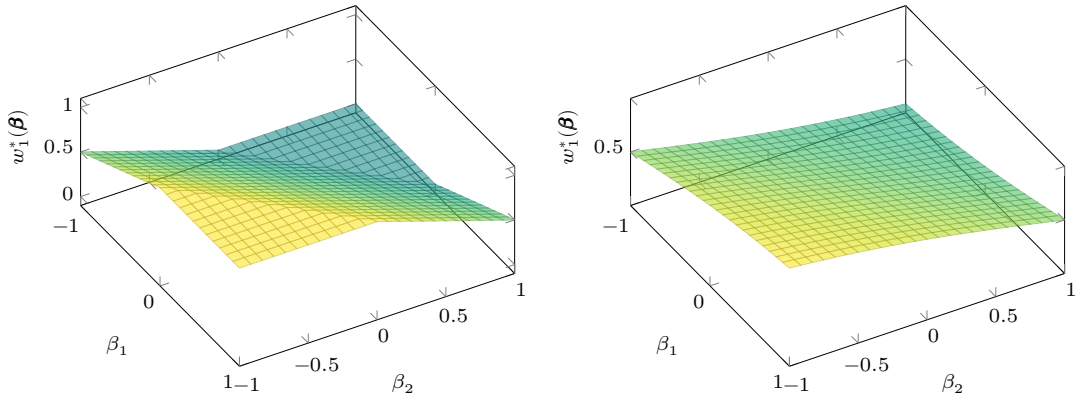
1. *For $\mathcal{W} = \left\{ \mathbf{w} \in \mathbb{R}^p \;\middle|\; \mathbf{w} \geq \mathbf{0}, \mathbf{1}^\top \mathbf{w} = 1 \right\}$:*

$$w_i^* = \frac{\alpha_i \exp(\beta_i)}{\sum_{j=1}^{p} \alpha_j \exp(\beta_j)} \quad \forall 1 \leq i \leq p. \tag{7.82}$$

2. *For $\mathcal{W} = \mathbb{R}^p_+$:*

$$w_i^* = \alpha_i \exp(\beta_i - 1) \quad \forall 1 \leq i \leq p. \tag{7.83}$$

An illustration comparing the differentiability of ADMM updates based on the Kullback-Leibler divergence and the standard Euclidean distance is given Figure 7.2.



$$(w_1^*, w_2^*) = \underset{\substack{\mathbf{w}\in\mathbb{R}_+^2 \\ \mathbf{1}^\top\mathbf{w}=1}}{\operatorname{argmin}} \left\{0.5\|\mathbf{w}-\mathbf{1}\|_2^2 - \boldsymbol{\beta}^\top\mathbf{w}\right\} \qquad (w_1^*, w_2^*) = \underset{\substack{\mathbf{w}\in\mathbb{R}_+^2 \\ \mathbf{1}^\top\mathbf{w}=1}}{\operatorname{argmin}} \left\{D_{\mathrm{KL}}(\mathbf{w},\mathbf{1}) - \boldsymbol{\beta}^\top\mathbf{w}\right\}$$

(a) Using Euclidean distance. (b) Using Kullback-Leibler divergence.

FIGURE 7.2 Differentiability of ADMM updates: comparison between the Kullback-Leibler divergence and the standard Euclidean distance. On the right-hand side we show an example of the first case of Lemma 7.1 (*c.f.* (7.82)) in two dimensions for the particular case where $\boldsymbol{\alpha} = \mathbf{1}$. On the left-hand side, we show the same results but replacing the Kullback-Leibler divergence by the Euclidean distance (*c.f.* Appendix C.1.1). One can observe that using the Kullback-Leibler divergence, the obtained solution is smooth, which is not the case when using the Euclidean distance.

As an example, consider the MAP inference problem with the following decomposition of the constraint sets:

$$\mathcal{X}_1 = \left\{\mathbf{x}\in\mathbb{R}^n \;\middle|\; \mathbf{x}\geq\mathbf{0},\ \text{sum of each row of mat}(\mathbf{x}) \text{ is } 1\right\}, \tag{7.84}$$

$$\mathcal{X}_2 = \left\{\mathbf{x}\in\mathbb{R}^n \;\middle|\; \mathbf{x}\geq\mathbf{0}\right\}. \tag{7.85}$$

Recall that mat$(\mathbf{x})$ denotes a reshaped version of $\mathbf{x}$ that represents the corresponding assignment matrix. In this case, mat$(\mathbf{x})$ is a $|\mathcal{V}|\times|\mathcal{S}|$ matrix, where $\mathcal{V}$ and $\mathcal{S}$ denote the set of nodes and the set of labels, and each row of mat$(\mathbf{x})$ corresponds to a node in the MRF.
Denote

$$\mathbf{v}^{(k+1)} = -\frac{1}{\rho}\left(\mathbf{u} + \frac{1}{2}\mathbf{P}\mathbf{z}^{(k)} + \mathbf{y}^{(k)}\right), \tag{7.86}$$

$$\mathbf{w}^{(k+1)} = -\frac{1}{\rho}\left(\frac{1}{2}\mathbf{P}^\top\mathbf{x}^{(k+1)} - \mathbf{y}^{(k)}\right), \tag{7.87}$$

then (7.76) and (7.77) become

$$\mathbf{x}^{(k+1)} = \underset{\mathbf{x}\in\mathcal{X}_1}{\operatorname{argmin}} \left\{D_{\mathrm{KL}}(\mathbf{x},\mathbf{z}^{(k)}) - \mathbf{v}^{(k+1)}\cdot\mathbf{x}\right\}, \tag{7.88}$$

$$\mathbf{z}^{(k+1)} = \underset{\mathbf{z}\in\mathcal{X}_2}{\operatorname{argmin}} \left\{D_{\mathrm{KL}}(\mathbf{z},\mathbf{x}^{(k+1)}) - \mathbf{w}^{(k+1)}\cdot\mathbf{z}\right\}. \tag{7.89}$$

It is easy to see that the minimization problem in (7.88) can be decomposed into subproblems at each row of $\text{mat}(\mathbf{x})$ whose solutions follow the first case in Lemma 7.1 (*i.e.* (7.82)), which is

$$X_{is}^{(k+1)} = \frac{Z_{is}^{(k)} \exp\left(V_{is}^{(k+1)}\right)}{\sum_{t \in \mathcal{S}} Z_{it}^{(k)} \exp\left(V_{it}^{(k+1)}\right)} \quad \forall s \in \mathcal{S}, \forall i \in \mathcal{V}, \tag{7.90}$$

where we have denoted $\mathbf{X} = \text{mat}(\mathbf{x}), \mathbf{Z} = \text{mat}(\mathbf{z})$ and $\mathbf{V} = \text{mat}(\mathbf{v})$.
For (7.89), again using the second case (since $\mathbf{z} \in \mathcal{X}_2 = \mathbb{R}_+^n$) of Lemma 7.1, we obtain

$$z_i^{(k+1)} = x_i^{(k+1)} \exp\left(w_i^{(k+1)} - 1\right), \quad \forall 1 \leq i \leq n. \tag{7.91}$$

Thanks to the use of the KL divergence, we have reduced the ADMM updates (7.76)–(7.77) to (7.91)–(7.90), which have analytic forms and are clearly differentiable. Hence, the conditions stated in Assumption 7.1 are satisfied and thus we can apply the gradient computation framework presented in Section 7.4. It should be noted that **these results are also valid for higher-order models**. Recall from Chapters 5 and 6 that the energy in these models can be written as a multi-linear function over decomposing blocks of variables, thus each update step of Bregman ADMM will consist of minimizing a sum of a KL divergence term and a linear term over one block of variables (while the other blocks are fixed), which is the same as (7.88) or (7.89).

7.6.3 *Gradient computation for Bregman ADMM*

Let us recall that the gradients

$$\left(\frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{z}^{(k)}}, \frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{y}^{(k)}}, \frac{\partial \mathbf{x}^{(k+1)}}{\partial \boldsymbol{\theta}}\right) \quad \text{and} \quad \left(\frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{x}^{(k+1)}}, \frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{y}^{(k)}}, \frac{\partial \mathbf{z}^{(k+1)}}{\partial \boldsymbol{\theta}}\right) \tag{7.92}$$

are necessary to run Algorithm 7.3 (forward mode) or Algorithm 7.4 (reverse mode) to compute the loss gradient $\nabla_{\boldsymbol{\theta}} L$. In principle, one can plug (7.86) and (7.87) into (7.90) and (7.91) to obtain analytic expressions of $\mathbf{x}^{(k+1)}$ (as a function of $\mathbf{z}^{(k)}, \mathbf{y}^{(k)}, \boldsymbol{\theta}$) and $\mathbf{z}^{(k+1)}$ (as a function of $\mathbf{x}^{(k+1)}, \mathbf{y}^{(k)}, \boldsymbol{\theta}$), which would result in analytic expressions of the derivatives in (7.92). In practice, however, these expressions may be very cumbersome and not easy to be implemented efficiently. The idea is to break down these complicated update steps into simpler operations in which it is much easier to compute derivatives. In the previous section we have seen that the $\mathbf{x}$ update (7.76) is the composition of (7.86) and (7.90), and the $\mathbf{z}$ update (7.77) is the composition of (7.87) and (7.91). Therefore, it is natural to introduce intermediate nodes $\mathbf{v}$ (7.86) and $\mathbf{w}$ (7.87) into our computational graph, as shown in Figure 7.3. Obviously one can further break down these operations into even smaller ones, *e.g.* (7.90) can be seen as a series of exponentiation, multiplication, addition and division. This is indeed the core idea behind automatic differentiation (see *e.g.* [Baydin et al., 2018]) where every node of the corresponding computational graph represents an elementary arithmetic operation or function. For our presentation, we decide not to go for finer grained level because of two reasons:[1] (1) we would like to keep our computational graph not too complex,

[1] In our experiments presented in Section 7.7.2, however, we take advantage of automatic differentiation implemented by deep learning libraries.

and (2) the current level already allows efficient and vectorized computation of the derivatives, as shown in the following.

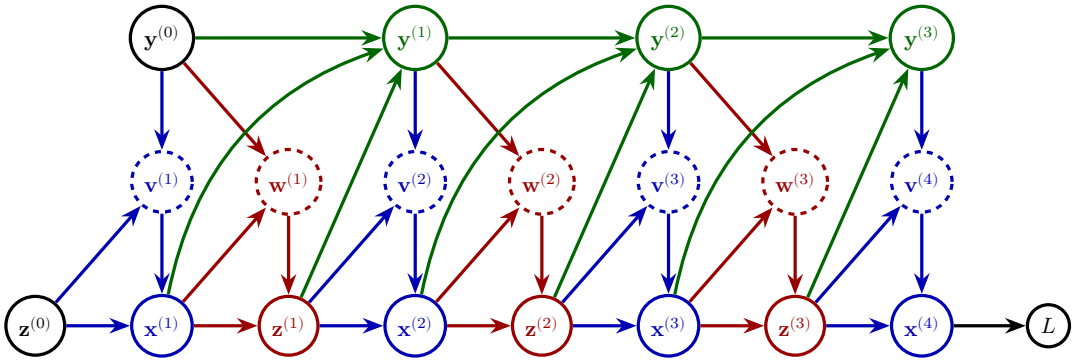


FIGURE 7.3 Computational graph illustrating ADMM for 4 iterations. Compared to the graph in Figure 7.1, here we have added intermediate nodes $\mathbf{v}^{(k)}$ and $\mathbf{w}^{(k)}$, defined by (7.86) and (7.87) for pairwise MAP inference or graph matching. A dashed circle represent a node that takes the parameter $\boldsymbol{\theta}$ as input, *e.g.* $\overset{\mathbf{v}}{(\quad)}$ means $\boldsymbol{\theta} \longrightarrow \overset{\mathbf{v}}{(\quad)}$. For clarity, we have not presented $\boldsymbol{\theta}$ explicitly.

In both forward-mode and reverse-mode differentiations (*c.f.* Sections 7.4.2 and 7.4.3, respectively), we need to compute the partial derivatives of every node with respect to its parents. For our KL-divergence Bregman ADMM, based on the computational graph in Figure 7.3, we can compute these derivatives as below. Without ambiguity, we omit the iteration counter k.

Derivatives of $\mathbf{x}^{(k+1)}$ with respect to $\mathbf{z}^{(k)}$ and $\mathbf{v}^{(k+1)}$

Denote $\mathbf{e} = \exp(\mathbf{v})$ (element-wise), $\mathbf{E} = \mathrm{mat}(\mathbf{e}), \mathbf{f} = \mathbf{z} \odot \mathbf{e}$ and

$$S_i = \sum_{t \in \mathcal{S}} Z_{it} E_{it} = \sum_{t \in \mathcal{S}} Z_{it} \exp(V_{it}) = \mathbf{1}^\top (\mathbf{z}_i \odot \mathbf{e}_i) \quad \forall i \in \mathcal{V}. \tag{7.93}$$

The Jacobian matrices $\frac{\partial \mathbf{x}}{\partial \mathbf{z}}$ and $\frac{\partial \mathbf{x}}{\partial \mathbf{v}}$ are block diagonal matrices whose diagonal blocks are respectively:

$$\frac{\partial \mathbf{x}_i}{\partial \mathbf{z}_i} = \frac{1}{S_i^2} \left(S_i \mathrm{diag}(\mathbf{e}_i) - \mathbf{e}_i \mathbf{e}_i^\top \mathrm{diag}(\mathbf{z}_i) \right) \qquad \forall i \in \mathcal{V}, \tag{7.94}$$

$$\frac{\partial \mathbf{x}_i}{\partial \mathbf{v}_i} = \frac{1}{S_i^2} \left(S_i \mathrm{diag}(\mathbf{f}_i) - \mathbf{f}_i \mathbf{f}_i^\top \right) \qquad \forall i \in \mathcal{V}. \tag{7.95}$$

A proof can be found in Appendix C.1.2. In practice, one should compute $\frac{\partial \mathbf{x}}{\partial \mathbf{z}}$ first and then update $\frac{\partial \mathbf{x}}{\partial \mathbf{v}}$ using

$$\frac{\partial \mathbf{x}_i}{\partial \mathbf{v}_i} = \mathrm{diag}(\mathbf{z}_i) \frac{\partial \mathbf{x}_i}{\partial \mathbf{z}_i} \quad \forall i \in \mathcal{V}. \tag{7.96}$$

Derivatives of $\mathbf{z}^{(k+1)}$ with respect to $\mathbf{x}^{(k+1)}$ and $\mathbf{w}^{(k+1)}$

From (7.91) it is straightforward that

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \operatorname{diag}(\exp(\mathbf{w} - \mathbf{1})), \qquad \frac{\partial \mathbf{z}}{\partial \mathbf{w}} = \operatorname{diag}(\mathbf{x} \odot \exp(\mathbf{w} - \mathbf{1})). \tag{7.97}$$

Derivatives of $\mathbf{v}^{(k+1)}$ with respect to $\mathbf{z}^{(k)}$, $\mathbf{y}^{(k)}$ and $\boldsymbol{\theta}$

From (7.86):

$$\frac{\partial \mathbf{v}}{\partial \mathbf{z}} = -\frac{1}{2\rho}\mathbf{P}^\top, \qquad \frac{\partial \mathbf{v}}{\partial \mathbf{y}} = -\frac{1}{\rho}\mathbf{I}, \qquad \frac{\partial \mathbf{v}}{\partial \boldsymbol{\theta}} = -\frac{1}{\rho}\left(\frac{\partial \mathbf{u}}{\partial \boldsymbol{\theta}} + \frac{1}{2}\frac{\partial \mathbf{P}}{\partial \boldsymbol{\theta}}\mathbf{z}\right). \tag{7.98}$$

If the parameters are independent between unary and pairwise potentials (which is usually the case), *e.g.* $\boldsymbol{\theta} := (\boldsymbol{\theta}_u, \boldsymbol{\theta}_p), \mathbf{u} := \mathbf{u}(\boldsymbol{\theta}_u), \mathbf{P} := \mathbf{P}(\boldsymbol{\theta}_p)$, then

$$\frac{\partial \mathbf{v}}{\partial \boldsymbol{\theta}_u} = -\frac{1}{\rho}\frac{\partial \mathbf{u}}{\partial \boldsymbol{\theta}_u}, \qquad \frac{\partial \mathbf{v}}{\partial \boldsymbol{\theta}_p} = -\frac{1}{2\rho}\frac{\partial \mathbf{P}}{\partial \boldsymbol{\theta}_p}\mathbf{z}. \tag{7.99}$$

As an example, in the semantic segmentation application presented in Section 7.7, we will see that the unary potentials $\mathbf{u}$ are obtained from a CNN and each element of the pairwise potentials $\mathbf{P}$ are defined by a linear combination of Gaussian kernels. In this case, $\boldsymbol{\theta}_u$ represents the set of weights of the CNN and $\boldsymbol{\theta}_p$ represents the parameters of the Gaussian kernels as well as their coefficients.

Derivatives of $\mathbf{w}^{(k+1)}$ with respect to $\mathbf{x}^{(k+1)}$, $\mathbf{y}^{(k)}$ and $\boldsymbol{\theta}$

From (7.87):

$$\frac{\partial \mathbf{w}}{\partial \mathbf{x}} = -\frac{1}{2\rho}\mathbf{P}, \qquad \frac{\partial \mathbf{w}}{\partial \mathbf{y}} = \frac{1}{\rho}\mathbf{I}, \qquad \frac{\partial \mathbf{w}}{\partial \boldsymbol{\theta}} = -\frac{1}{2\rho}\frac{\partial \mathbf{P}^\top}{\partial \boldsymbol{\theta}}\mathbf{x}. \tag{7.100}$$

Similarly to the previous case, if $\boldsymbol{\theta}$ are decomposed into unary and pairwise parameters $(\boldsymbol{\theta}_u, \boldsymbol{\theta}_p)$ then

$$\frac{\partial \mathbf{w}}{\partial \boldsymbol{\theta}_u} = \mathbf{0}, \qquad \frac{\partial \mathbf{v}}{\partial \boldsymbol{\theta}_p} = -\frac{1}{2\rho}\frac{\partial \mathbf{P}^\top}{\partial \boldsymbol{\theta}_p}\mathbf{x}. \tag{7.101}$$

Derivatives of $\mathbf{y}^{(k+1)}$ with respect to $\mathbf{y}^{(k)}$, $\mathbf{x}^{(k+1)}$ and $\mathbf{z}^{(k+1)}$

From (7.78):

$$\frac{\partial \mathbf{y}^{(k+1)}}{\partial \mathbf{y}^{(k)}} = \mathbf{I}, \qquad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \rho\mathbf{I}, \qquad \frac{\partial \mathbf{y}}{\partial \mathbf{z}} = -\rho\mathbf{I}, \tag{7.102}$$

where in the first equation we have kept the iteration counter to distinguish between $\mathbf{y}^{(k+1)}$ and $\mathbf{y}^{(k)}$.

Finally, we present a complete Bregman ADMM algorithm for solving pairwise MAP inference with reverse-mode differentiation in Algorithm 7.5. Forward-mode differentiation can be derived in a straightforward manner, but as argued in Section 7.4.4, reverse-mode is more efficient for our training problem since the output of our computational graph is a scalar while the dimensions of the inputs are large.

ALGORITHM 7.5 Bregman ADMM for pairwise MAP inference with reverse-mode differentiation, corresponding to the computational graph in Figure 7.3.

Forward pass:

1: Initialize $\mathbf{z}_0, \mathbf{y}_0$.
2: **for** $k = 0, 1, \dots, N-1$ **do**
3: Compute $\mathbf{v}^{(k+1)}$ using (7.86).
4: Compute and store in the memory $\frac{\partial \mathbf{v}^{(k+1)}}{\partial \mathbf{z}^{(k)}}, \frac{\partial \mathbf{v}^{(k+1)}}{\partial \mathbf{y}^{(k)}}, \frac{\partial \mathbf{v}^{(k+1)}}{\partial \boldsymbol{\theta}}$ using (7.98).
5: Compute $\mathbf{x}^{(k+1)}$ using (7.90).
6: Compute and store in the memory $\frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{z}^{(k)}}, \frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{v}^{(k+1)}}$ using (7.94) and (7.95).
7: Compute $\mathbf{w}^{(k+1)}$ using (7.87).
8: Compute and store in the memory $\frac{\partial \mathbf{w}^{(k+1)}}{\partial \mathbf{x}^{(k+1)}}, \frac{\partial \mathbf{w}^{(k+1)}}{\partial \mathbf{y}^{(k)}}, \frac{\partial \mathbf{w}^{(k+1)}}{\partial \boldsymbol{\theta}}$ using (7.100).
9: Compute $\mathbf{z}^{(k+1)}$ using (7.91).
10: Compute and store in the memory $\frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{x}^{(k+1)}}, \frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{w}^{(k+1)}}$ using (7.97).
11: Compute $\mathbf{y}^{(k+1)}$ using (7.78).
12: **end for**
13: Compute the loss $L(\mathbf{x}^{(N)})$ and its derivative $\frac{\partial L}{\partial \mathbf{x}^{(N)}}$.

Reverse pass:

1: Initialization:

$$\bar{\mathbf{x}}^{(N)} \leftarrow \frac{\partial L}{\partial \mathbf{x}^{(N)}}, \ \bar{\mathbf{v}}^{(N)} \leftarrow \frac{\partial \mathbf{x}^{(N)}}{\partial \mathbf{v}^{(N)}} \bar{\mathbf{x}}^{(N)}, \ \bar{\mathbf{y}}^{(N-1)} \leftarrow -\frac{1}{\rho} \bar{\mathbf{v}}^{(N)}, \ \bar{\boldsymbol{\theta}} \leftarrow \frac{\partial \mathbf{v}^{(N)}}{\partial \boldsymbol{\theta}} \bar{\mathbf{v}}^{(N)}.$$

2: **for** $k = (n-1), (n-2), \dots, 1$ **do**

$$\bar{\mathbf{z}}^{(k)} \leftarrow \frac{\partial \mathbf{x}^{(k+1)}}{\partial \mathbf{z}^{(k)}} \bar{\mathbf{x}}^{(k+1)} - \frac{1}{2\rho} \mathbf{P}^\top \bar{\mathbf{v}}^{(k+1)} - \rho \bar{\mathbf{y}}^{(k)}, \tag{7.103}$$

$$\bar{\mathbf{w}}^{(k)} \leftarrow \frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{w}^{(k)}} \bar{\mathbf{z}}^{(k)}, \tag{7.104}$$

$$\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \frac{\partial \mathbf{w}^{(k)}}{\partial \boldsymbol{\theta}} \bar{\mathbf{w}}^{(k)}, \tag{7.105}$$

$$\bar{\mathbf{x}}^{(k)} \leftarrow \frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{x}^{(k)}} \bar{\mathbf{z}}^{(k)} - \frac{1}{2\rho} \mathbf{P} \bar{\mathbf{w}}^{(k)} + \rho \bar{\mathbf{y}}^{(k)}, \tag{7.106}$$

$$\bar{\mathbf{v}}^{(k)} \leftarrow \frac{\partial \mathbf{x}^{(k)}}{\partial \mathbf{v}^{(k)}} \bar{\mathbf{v}}^{(k)}, \tag{7.107}$$

$$\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \frac{\partial \mathbf{v}^{(k)}}{\partial \boldsymbol{\theta}} \bar{\mathbf{v}}^{(k)}, \tag{7.108}$$

$$\bar{\mathbf{y}}^{(k-1)} \leftarrow \bar{\mathbf{y}}^{(k)} - \frac{1}{\rho} \bar{\mathbf{v}}^{(k)} + \frac{1}{\rho} \bar{\mathbf{w}}^{(k)}. \tag{7.109}$$

3: **end for**
4: Return $\nabla_{\boldsymbol{\theta}} L := \bar{\boldsymbol{\theta}}$.

7.7 APPLICATION: DENSE CRFS FOR SEMANTIC SEGMENTATION

7.7.1 *Semantic segmentation and dense CRFs*

(a) `person`, `car`, `horse` and `background`. (b) `person`, `bicycle` and `background`.

FIGURE 7.4 Semantic segmentation consists of assigning each image pixel to an object class. Images and annotations taken from the Pascal VOC 2012 dataset [Everingham et al., 2010]. Best viewed in color.

Semantic segmentation consists of assigning each image pixel to an object class (*c.f.* Figure 7.4). This problem can be easily modeled by MAP inference on a CRF, where each node corresponds to a pixel on the image, and each label corresponds to an object class.

Indeed, consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ defined over an image where each node in $\mathcal{V}$ corresponds to a pixel. Let $\mathcal{S}$ be the set of labels (*i.e.* object classes in our case). With suitable potential functions, the segmentation problem can be reduced to minimizing a CRF energy:

$$\min_{\mathbf{s} \in \mathcal{S}^{|\mathcal{V}|}} e(\mathbf{s}) = \sum_{i \in \mathcal{V}} \psi_i(s_i) + \sum_{ij \in \mathcal{E}} \psi_{ij}(s_i, s_j), \tag{7.110}$$

where $\mathbf{s} = (s_i)_{i \in \mathcal{V}}$ denotes the joint labeling of all pixels. As usual, we re-write the above problem using indicator functions:

$$\begin{aligned} \min \quad & E(\mathbf{x}) = \sum_{i \in \mathcal{V}} \sum_{s \in \mathcal{S}} \psi_i(s) x_i(s) + \sum_{ij \in \mathcal{E}} \sum_{s \in \mathcal{S}} \sum_{t \in \mathcal{S}} \psi_{ij}(s,t) x_i(s) x_j(t), \\ \text{s.t.} \quad & \sum_{s \in \mathcal{S}} x_i(s) = 1 \quad \forall i \in \mathcal{V}, \\ & x_i(s) \in \{0, 1\} \quad \forall s \in \mathcal{S}, \forall i \in \mathcal{V}. \end{aligned} \tag{7.111}$$

The continuous relaxation of this problem (*i.e.* when $x_i(s) \in \{0, 1\}$ is replaced by $x_i(s) \geq 0$) can be re-written in using vector notation as:

$$\begin{aligned} \min \quad & E(\mathbf{x}) = \mathbf{u}^\top \mathbf{x} + \frac{1}{2} \mathbf{x}^\top \mathbf{P} \mathbf{x}, \\ \text{s.t.} \quad & \mathbf{1}^\top \mathbf{x}_i = 1 \quad \forall i \in \mathcal{V}, \\ & \mathbf{x}_i \geq \mathbf{0} \quad \forall i \in \mathcal{V}, \end{aligned} \tag{7.112}$$

where $\mathbf{u}$ and $\mathbf{P}$ represent the potentials, $\mathbf{u} \in \mathbb{R}^n, n = |\mathcal{V}||\mathcal{S}|, \mathbf{P} \in \mathbb{R}^{n \times n}$.

The unary potentials of this model can be obtained from some pixel-level classifier such as k-means clustering, Gaussian mixture model, or a neural network for example. The edges and their potentials should be defined in such a way that pixels of the same object tend to have the same label. The connectivities between the nodes are often defined over neighboring ones only (*i.e.* $\mathbf{P}$ is very sparse) since high connectivities imply high computational cost. This, however, limits the expressive power of the model since two distant pixels might have a strong effect on each other, yet this relationship is not taken into account by a sparse model. To overcome this issue, [Krähenbühl and Koltun, 2011] proposed in their influential work a *dense* (or fully connected) CRF with Gaussian pairwise potentials that take the form

$$\psi_{ij}(s,t) = \sum_{c=1}^{C} \mu_c(s,t) k_c(\mathbf{f}_i, \mathbf{f}_j), \tag{7.113}$$

where $k_c(\mathbf{f}_i, \mathbf{f}_j) = \exp\left(-\frac{1}{2}(\mathbf{f}_i - \mathbf{f}_j)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{f}_i - \mathbf{f}_j)\right)$ is a Gaussian kernel and μ_c is called a *compatibility function*. A simple compatibility function is the Potts model: $\mu_c(s,t) = w_c \mathbb{1}_{[s \neq t]}$. For image segmentation, [Krähenbühl and Koltun, 2011] proposed a contrast-sensitive two-kernel potential function:

$$\psi_{ij}(s,t) = \mu_1(s,t) \exp\left(-\frac{\|p_i - p_j\|_2^2}{2\theta_\alpha^2} - \frac{\|I_i - I_j\|_2^2}{2\theta_\beta^2}\right) + \mu_2(s,t) \exp\left(-\frac{\|p_i - p_j\|_2^2}{2\theta_\gamma^2}\right), \tag{7.114}$$

where p_i, p_j denotes the positions and I_i, I_j the colors of the pixels i, j respectively.

To minimize the CRF energy with these Gaussian potentials, the corresponding authors used a *mean field* approximation to the CRF distribution. They showed that a mean field update of all variables in a dense CRF can be performed very efficiently using Gaussian filtering in the feature space. The resulted algorithm is very efficient and is able to capture fine edge details while also catering for long range dependencies. We refer to [Krähenbühl and Koltun, 2011] for further details.

Since the publication of [Krähenbühl and Koltun, 2011], mean field dense CRFs have been used as a post-processing step for pixel-level classifiers such as deep CNNs and consistently yielded state-of-the-art results (see *e.g.* [Chen et al., 2014]). In these works, the parameters of the pixel-level classifier and of the CRF were learned separately, which might be suboptimal because the two models are unaware of each other during training. To address this issue, [Krähenbühl and Koltun, 2013] — inspired by [Domke, 2012] — proposed a method for training jointly the pixel-level classifier and the CRF in an end-to-end manner using unrolled optimization (mean field) and reverse-mode differentiation (*c.f.* Section 7.4). The experiments therein showed a substantial improvement over training separately a CRF and a TextonBoost classifier [Shotton et al., 2009]. Later, [Zheng et al., 2015] applied the same idea but with a fully convolutional neural network classifier [Long et al., 2015] and achieved state-of-the-art results at the time of publication.

Our work presented in this section can be considered to be in the same line of work of [Krähenbühl and Koltun, 2013] and [Zheng et al., 2015], in the sense that all works use an unrolled optimization algorithm for minimizing the CRF energy, enabling back-propagation of the gradient through the CRF, which allows training it jointly and

end-to-end with a pixel-level classifier. An illustration of this idea is given in Figure 7.5. The major difference between our work and the others is that we use Bregman ADMM to optimize the CRF (*c.f.* Section 7.6.2), whereas they used mean field inference. The experiments in the next section show that our method achieve better results.

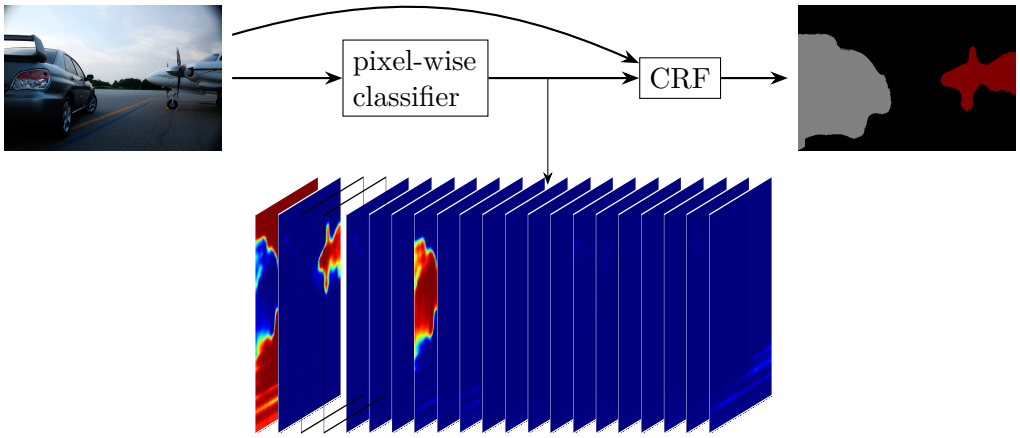


FIGURE 7.5 Illustration of a single network composed of a pixel-wise classifier and a CRF for semantic image segmentation. The output of the pixel-wise classifier is fed into the CRF as its unary potentials. If CRF inference consists of a sequence of differentiable operations, then this combined network can be trained jointly in an end-to-end manner using *back-propagation* (or *reverse-mode differentiation*). This idea was introduced in [Krähenbühl and Koltun, 2013].

7.7.2 Experiments

We performed experiments on the Pascal VOC 2012 dataset [Everingham et al., 2010], a commonly used benchmark for comparing semantic segmentation algorithms.

Dataset

As a standard practice, we augmented the Pascal VOC 2012 dataset with images from the dataset of [Hariharan et al., 2014]. Our training set contains in total 11685 images, which consists of the training ones from Pascal VOC 2012 (1464 images), plus the training and validation ones from [Hariharan et al., 2014]. After removing the overlap between Pascal VOC 2012 validation images and our training set, we were left with 346 images from the original Pascal VOC 2012 validation set. The methods were also evaluated on the Pascal VOC 2012 *test* set (1457 images) whose annotations are not publicly available.

Models

Our model consists of a CNN, followed by a dense CRF (*c.f.* Figure 7.5). For the CNN part, following [Zheng et al., 2015], we used the fully convolutional network (**FCN**) architecture of [Long et al., 2015]. In addition, we considered also an improved version of this network introduced in [Chen et al., 2014] that uses 'atrous' convolution. Let us denote this atrous version **AFCN**. For the CRF part, we used the CRF layer proposed by [Zheng et al., 2015], which is a stack of mean field iterations. On the other hand, our proposed CRF layer is a stack of Bregman ADMM iterations. Let us denote

these models respectively *mean field CRF* (**MFCRF**) and *alternating direction CRF* (**ADCRF**). Our main goal is to compare ADCRF against MFCRF.

Implementation details

We implemented our model on top of the publicly available code of [Zheng et al., 2015][2] and [Monteiro et al., 2018][3], which are based on the popular deep learning library Keras [Chollet et al., 2015]. All models were trained on an NVIDIA GeForce GTX 1080 Ti, with image sizes 500×500 and batch size 4.

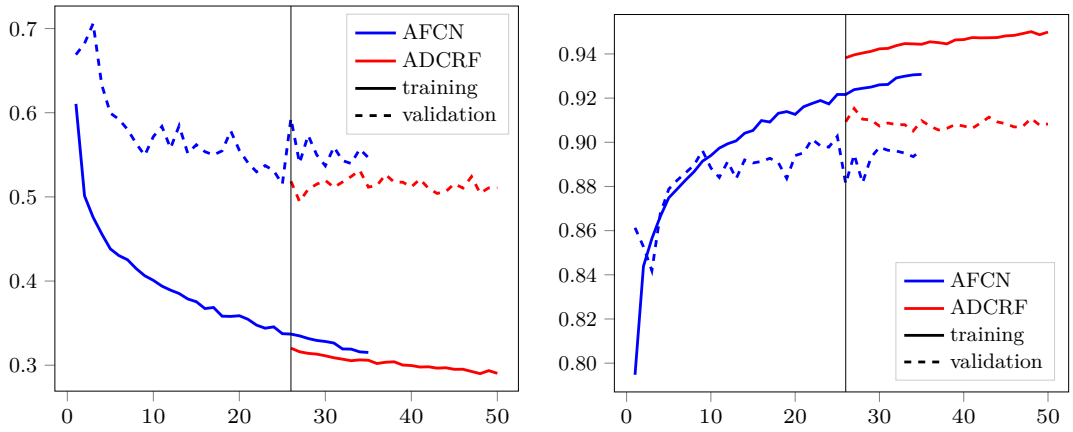
We first trained the CNN part to convergence (that is when overfitting starts to occur). Using a learning rate of 10^{-5}, this training took about 25 epochs.[4] Then we used this trained model as initialization for training end-to-end the CNN+CRF network using the same learning rate of 10^{-5}. Following [Zheng et al., 2015], we set the number of CRF iterations to 5. For ADCRF we set the penalty parameter ρ to 1.0. The CRF compatibility parameters are initialized using the Potts model, and the CRF kernel parameters are obtained from [Zheng et al., 2015].

Results

First, as an important validation step, we demonstrate that our ADCRF layer can effectively back-propagate the loss gradient, allowing successful end-to-end training. Indeed, we present in Figure 7.6 the loss value and the pixel accuracy[5] per training epoch of AFCN alone versus AFCN+ADCRF. In this experiment, we trained AFCN+ADCRF using as initialization the parameters of AFCN trained for 25 epochs (this can be understood as training AFCN for 25 epochs then plugging in the CRF and continue training). One can observe that the loss in AFCN+ADCRF consistently decreases, which confirms that gradient based learning through ADCRF was successful. Moreover, one can also observe a dramatic improvement obtained by adding ADCRF, in both training and validation loss as well as accuracy, compared to continuing training AFCN alone.

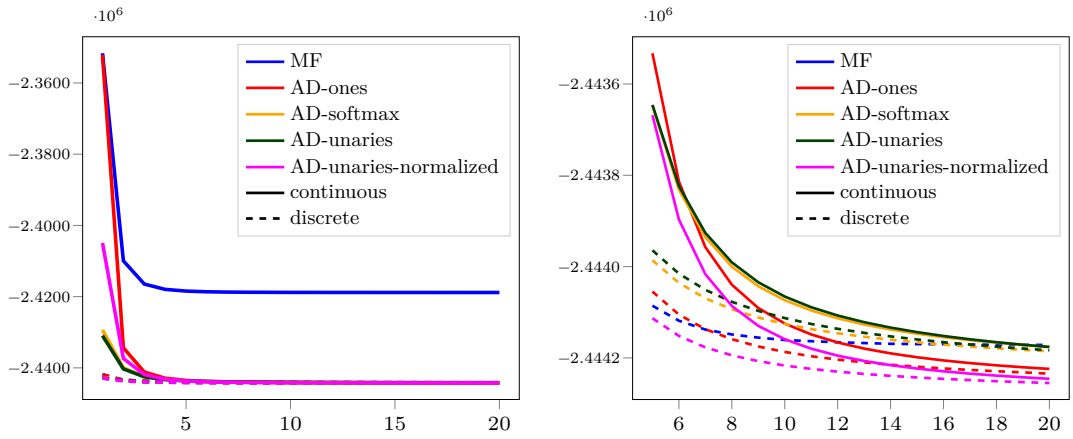
Second, let us recall that the original motivation of using nonconvex ADMM was because of its great inference performance, in terms of energy minimization. We performed an experiment to validate this claim as well. We compare the energy values produced by mean-field versus ADMM on the 1456 images of the Pascal VOC 2012 test set, using the trained AFCN+MFCRF model. This means that the training phase used mean-field while in the inference phase we use both mean-field and ADMM for comparison. Note that here we have given an advantage to mean-field. The obtained results are given in Figure 7.7. One can observe that ADMM clearly outperformed mean-field in terms of continuous energy minimization. After rounding to obtain discrete solutions, ADMM with any initialization can outperform mean-field given enough iterations. In particular, when the output of the CNN, normalized to $[-1, 1]$, is used for initialization, ADMM always achieves lower discrete energy values than mean-field (with the same number of iterations). In our experiments we used this initialization.

[2] https://github.com/sadeepj/crfasrnn_keras
[3] https://github.com/MiguelMonteiro/permutohedral_lattice
[4] An epoch is a complete pass through the entire training set.
[5] Pixel accuracy represents the percentage of pixels that are correctly classified.



(a) Loss per training epoch. (b) Pixel accuracy per training epoch.

FIGURE 7.6 Loss value and pixel accuracy per training epoch of AFCN alone versus AFCN+ADCRF. The AFCN alone network achieved the best accuracy on the validation set at epoch 25. We used the trained AFCN parameters obtained at this epoch as initialization for training AFCN+ADCRF. One can observe a dramatic improvement obtained adding ADCRF, compared to continuing training AFCN alone.



(a) Mean energy per iteration. (b) Zoomed version of the plot on the left.

FIGURE 7.7 Mean energy values of mean-field inference versus alternating direction inference on the 1456 images of the Pascal VOC 2012 test set, using a trained AFCN+MFCRF network. Results for ADCRF are presented for different initializations. The right hand-side graph is a zoomed version of the left hand-side one. It is observed that ADMM clearly outperformed mean-field inference in terms of continuous energy minimization. After rounding to obtain discrete solutions, ADMM with any initialization can outperform mean-field given enough iterations. When the normalized output of the CNN is used for initialization, ADMM always achieves lower discrete energy than mean-field with the same number of iterations.

Finally, we report the accuracy, in terms of the mean intersection over union (mIOU) score, of the models on the validation set and the test set in Table 7.2. The detailed results for each object class are presented in Appendix C.

While the number of CRF iterations was set to 5 for training, it was observed in [Zheng et al., 2015] that setting this value to 10 for inference yields better results. Since mean-field generally converges within 10 iterations (as observed previ-

TABLE 7.2 Accuracy of the models on the Pascal VOC 2012 segmentation dataset.

Model	Validation mIOU	Test mIOU
FCN	64.3724	67.0110
FCN+MFCRF	66.2431	68.9902
FCN+ADCRF10	66.6076	70.1612
FCN+ADCRF50	**66.7832**	**70.3331**
AFCN	65.0996	67.7054
AFCN+MFCRF	68.2995	70.5693
AFCN+ADCRF10	70.0467	70.9005
AFCN+ADCRF50	**70.2820**	**71.0304**

ously by [Krähenbühl and Koltun, 2013, Zheng et al., 2015] and again by our results in Figure 7.7), setting the number of CRF iterations to a higher value will not yield significant improvement. On the contrary, as Figure 7.7 indicates, ADMM only converges after a much higher number of iterations. Therefore, in Table 7.2 we also report the results for 10 and 50 ADCRF iterations.

We have a number of observations. First, adding a CRF layer clearly improved the accuracy over plain CNN. Second, ADCRF outperformed MFCRF, both on the validation and the test sets. Third, ADCRF with 50 iterations produced the best results. To some extent, the fact that ADMM does not converge after 10 iterations (yet it is still better than MFCRF) is not a bad thing, because it leaves the user with a larger range of trade-off between accuracy and computational time. It should be noted that we do not have such choice with MFCRF.

We conclude this section with some qualitative results in Figure 7.8.

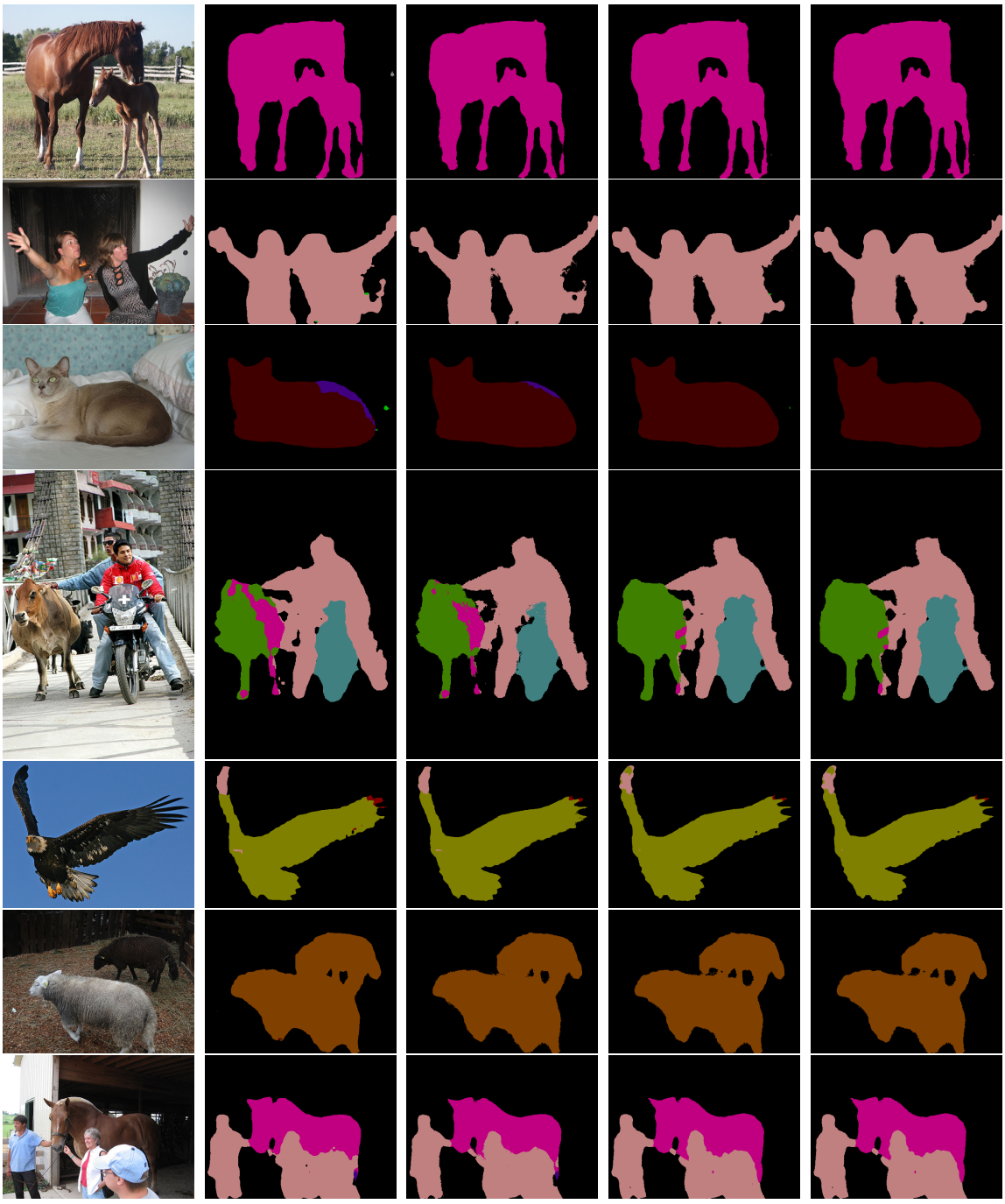


FIGURE 7.8 Qualitative results on the Pascal VOC 2012 test set. From left to right: input image, FCN, FCN + MFCRF (10 iterations), FCN + ADCRF (10 iterations), FCN + ADCRF (50 iterations).

8

Discussion & Conclusion

In this thesis, we have presented our contributions to graph-based representations in computer vision, in terms of both inference and learning.

First, we proposed Alternating Direction Graph Matching (ADGM), a novel decomposition framework for solving graph and hypergraph matching based on nonconvex ADMM. This framework is very general and includes an infinite number of algorithms that can be applied to models with arbitrary potential functions as well as arbitrary matching constraints. We implemented two instantiations of this framework and evaluated them against existing methods on popular datasets. The results showed that our algorithms achieved state-of-the-art performance.

Second, we proposed a nonconvex continuous relaxation of MAP inference in arbitrary Markov random fields. This relaxation was shown to be tight, *i.e.* equivalent to the original discrete problem. For solving this continuous relaxation, we presented solutions using two popular gradient-based methods, and further introduced a more effective solution by nonconvex ADMM. Experiments on different real-world problems demonstrate that the proposed ADMM compares favorably with state-of-the-art algorithms in different settings.

Third, we proposed a method for learning the parameters of these graph-based models from training data, based on nonconvex ADMM. This method consists of viewing ADMM iterations as a sequence of differentiable operations, which allows efficient computation of the gradient of the training loss with respect to the model parameters, enabling efficient training using stochastic gradient descent. We presented experiments on a popular semantic segmentation dataset, which demonstrated that our method has better performance than the current state-of-the-art mean-field based algorithm.

A number of questions and a fair amount of research remain open for future work.

The proposed ADMM frameworks can have infinitely many instantiations (*i.e.* many *decompositions*), as discussed in Sections 5.2 and 6.4.2. A natural question arises: What is the best decomposition? There is probably no single best decomposition for all problems, but for a specific one there might be decompositions that are better than others.

Unlike for convex problems, the solution quality as well as the convergence speed of nonconvex ADMM are very sensitive to the parameters, especially the penalty ρ, as discussed in Sections 5.3.5 and 6.4.2. What are the best values for these parameters? The theoretical convergence of the proposed algorithms also needs to be completed.

The proposed nonconvex continuous relaxation of MAP inference is provably tight,

as shown in Section 6.3. Does it hold for the continuous relaxation of graph matching?

In the learning framework presented in Chapter 7, the penalty parameter ρ is fixed (and set to 1.0 in the experiments). What would be the best value of ρ? It's also worth investigating whether allowing ρ to change between iterations (as done for the inference tasks in Chapters 5 and 6) could yield better results.

Also in Chapter 7, we only gave an example of penalty functions that allow differentiable ADMM updates, which is the Kullback-Leibler divergence. It's worth finding and evaluating other penalty functions as well.

These are interesting research directions that we would like to follow in the future.

A

Theoretical Proofs and Additional Experimental Results for Chapter 5

We give proofs of the theoretical results presented in Chapter 5 in Section A.1. Additional experimental results are provided in Section A.2.

A.1 PROOFS OF THEORETICAL RESULTS

A.1.1 *Proof of Equations (5.35), (5.36) and (5.38)*

We showed in Section 5.3 how the subproblems in **ADGM1** algorithm can be reduced to (5.32) where $(\mathbf{c}_d)_{1\leq d\leq D}$ are defined by (5.33) and (5.34). In this appendix, we show how the subproblems in **ADGM2** can be reduced to (5.32) where $(\mathbf{c}_d)_{1\leq d\leq D}$ are defined by (5.35) (5.36) and (5.38).

Recall that in ADGM2, we chose $(\mathbf{A}_d)_{1\leq d\leq D}$ such that

$$\mathbf{x}_1 = \mathbf{x}_2, \quad \mathbf{x}_2 = \mathbf{x}_3, \dots, \quad \mathbf{x}_{D-1} = \mathbf{x}_D, \tag{A.1}$$

which can be written in matrix form as follows:

$$\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} -\mathbf{x}_2 \\ \mathbf{x}_2 \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ -\mathbf{x}_3 \\ \mathbf{x}_3 \\ \vdots \\ \mathbf{0} \end{bmatrix} + \cdots + \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ -\mathbf{x}_D \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{A.2}$$

The above can be in turn re-written as $\mathbf{A}_1\mathbf{x}_1 + \mathbf{A}_2\mathbf{x}_2 + \cdots + \mathbf{A}_D\mathbf{x}_D = \mathbf{0}$ where $\mathbf{A}_d$ is chosen to be the d^{th} (block) **column** of the following $(D-1)\times D$ block matrix $\mathbf{A}$ and $\mathbf{y}$ is also a $(D-1)\times 1$ block vector:

$$\mathbf{A} = \begin{bmatrix} \mathbf{I} & -\mathbf{I} & & & & \\ & \mathbf{I} & -\mathbf{I} & & & \\ & & \mathbf{I} & \ddots & & \\ & & & \ddots & -\mathbf{I} & \\ & & & & \mathbf{I} & -\mathbf{I} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} \mathbf{y}_2 \\ \mathbf{y}_3 \\ \vdots \\ \mathbf{y}_D \end{bmatrix}. \tag{A.3}$$

From (5.20) we easily have

$$\mathbf{s}_1^{(k)} = \begin{bmatrix} \mathbf{0} - \mathbf{x}_2^{(k)} \\ \mathbf{x}_2^{(k)} - \mathbf{x}_3^{(k)} \\ \mathbf{x}_3^{(k)} - \mathbf{x}_4^{(k)} \\ \vdots \\ \mathbf{x}_{D-1}^{(k)} - \mathbf{x}_D^{(k)} \end{bmatrix}, \quad \mathbf{s}_D^{(k)} = \begin{bmatrix} \mathbf{x}_1^{(k+1)} - \mathbf{x}_2^{(k+1)} \\ \mathbf{x}_2^{(k+1)} - \mathbf{x}_2^{(k+1)} \\ \vdots \\ \mathbf{x}_{D-2}^{(k+1)} - \mathbf{x}_{D-1}^{(k+1)} \\ \mathbf{x}_{D-1}^{(k+1)} - \mathbf{0} \end{bmatrix} \tag{A.4}$$

and

$$\mathbf{s}_d^{(k)} = \begin{bmatrix} \mathbf{x}_1^{(k+1)} - \mathbf{x}_2^{(k+1)} \\ \vdots \\ \mathbf{x}_{d-2}^{(k+1)} - \mathbf{x}_{d-1}^{(k+1)} \\ \mathbf{x}_{d-1}^{(k+1)} - \mathbf{0} \\ \mathbf{0} - \mathbf{x}_{d+1}^{(k)} \\ \mathbf{x}_{d+1}^{(k)} - \mathbf{x}_{d+2}^{(k)} \\ \vdots \\ \mathbf{x}_{D-1}^{(k)} - \mathbf{x}_D^{(k)} \end{bmatrix} \quad \forall 2 \leq d \leq D-1. \tag{A.5}$$

Now we compute the vectors $(\mathbf{c}_d)_{1 \leq d \leq D}$.

- For $d = 1$: Since $\mathbf{A}_1 = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix}^\top$ we have

$$\mathbf{A}_1^\top \mathbf{A}_1 = \mathbf{I}, \tag{A.6}$$
$$\mathbf{A}_1^\top \mathbf{s}_1^{(k)} = -\mathbf{x}_2^{(k)}, \tag{A.7}$$
$$\mathbf{A}_1^\top \mathbf{y}^{(k)} = \mathbf{y}_2^{(k)}. \tag{A.8}$$

  Plugging these into (5.22), it becomes

$$\frac{1}{2}\|\mathbf{x}\|_2^2 + \left(-\mathbf{x}_2^{(k)} + \frac{1}{\rho}\mathbf{y}_2^{(k)} + \frac{1}{\rho}\mathbf{p}_1^{(k)}\right)^\top \mathbf{x}. \tag{A.9}$$

  Clearly, minimizing this quantity over $\mathcal{M}_1$ is equivalent to solving (5.32) for $d = 1$, where $\mathbf{c}_1$ is defined by (5.35).

- For $d = D$: Since $\mathbf{A}_D = \begin{bmatrix} \mathbf{0} & \cdots & \mathbf{0} & -\mathbf{I} \end{bmatrix}^\top$ we have

$$\mathbf{A}_D^\top \mathbf{A}_D = \mathbf{I}, \tag{A.10}$$
$$\mathbf{A}_D^\top \mathbf{s}_D^{(k)} = -\mathbf{x}_{D-1}^{(k+1)}, \tag{A.11}$$
$$\mathbf{A}_D^\top \mathbf{y}^{(k)} = -\mathbf{y}_D^{(k)}. \tag{A.12}$$

  Plugging these into (5.22), it becomes

$$\frac{1}{2}\|\mathbf{x}\|_2^2 + \left(-\mathbf{x}_{D-1}^{(k)} - \frac{1}{\rho}\mathbf{y}_D^{(k)} + \frac{1}{\rho}\mathbf{p}_D^{(k)}\right)^\top \mathbf{x}. \tag{A.13}$$

  Minimizing this quantity over $\mathcal{M}_D$ is equivalent to solving (5.32) for $d = D$, where $\mathbf{c}_D$ is defined by (5.36).

- For $2 \leq d \leq D-1$: Since (the below non-zero blocks are at the $(d-1)$-th and d-th positions)

$$\mathbf{A}_d = \begin{bmatrix} \mathbf{0} & \cdots & \mathbf{0} & -\mathbf{I} & \mathbf{I} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix}^\top$$

we have

$$\begin{aligned} \mathbf{A}_d^\top \mathbf{A}_d &= 2\mathbf{I}, \\ \mathbf{A}_d^\top \mathbf{s}_d^{(k)} &= -\mathbf{x}_{d-1}^{(k+1)} - \mathbf{x}_{d+1}^{(k)}, \\ \mathbf{A}_d^\top \mathbf{y}^{(k)} &= -\mathbf{y}_d^{(k)} + \mathbf{y}_{d+1}^{(k)}. \end{aligned}$$

Plugging these into (5.22), it becomes

$$\|\mathbf{x}\|_2^2 + \left(-\mathbf{x}_{d-1}^{(k+1)} - \mathbf{x}_{d+1}^{(k)} - \frac{1}{\rho}(\mathbf{y}_d^{(k)} - \mathbf{y}_{d+1}^{(k)}) + \frac{1}{\rho}\mathbf{p}_d^{(k)}\right)^\top \mathbf{x}. \tag{A.14}$$

Minimizing this quantity over $\mathcal{M}_d$ is equivalent to solving (5.32), where $\mathbf{c}_d$ is defined by (5.38).

A.1.2 *Proof of Lemma 5.1*

For part (a), see for example [Condat, 2016]. For part (b), the corresponding KKT conditions are:

$$\begin{aligned} \mathbf{u} &\geq \mathbf{0}, && \text{(A.15)} \\ \mathbf{1}^\top \mathbf{u} &\leq 1, && \text{(A.16)} \\ \boldsymbol{\mu} &\geq \mathbf{0}, && \text{(A.17)} \\ \nu &\geq 0, && \text{(A.18)} \\ \mu_i u_i &= 0 \quad \forall 1 \leq i \leq d, && \text{(A.19)} \\ \nu(\mathbf{1}^\top \mathbf{u} - 1) &= 0, && \text{(A.20)} \\ u_i - c_i + \nu - \mu_i &= 0 \quad \forall 1 \leq i \leq d. && \text{(A.21)} \end{aligned}$$

If $\nu = 0$ then from (A.15), (A.17), (A.19) and (A.21) we have

$$\begin{aligned} u_i &\geq 0 \quad \forall i, && \text{(A.22)} \\ u_i - c_i = \mu_i &\geq 0 \quad \forall i, && \text{(A.23)} \\ u_i(u_i - c_i) &= 0 \quad \forall i, && \text{(A.24)} \end{aligned}$$

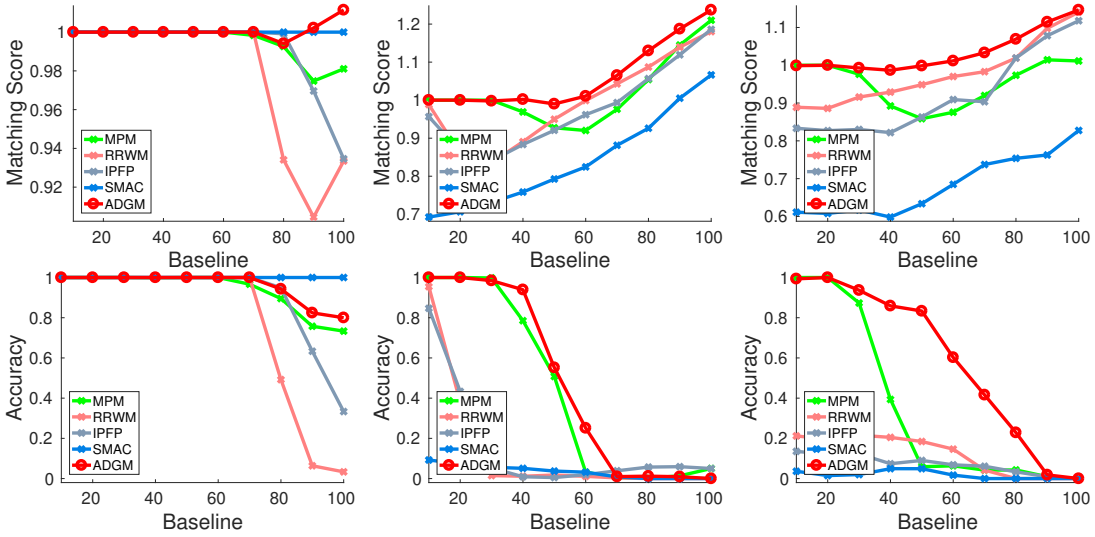
which yields $\mathbf{u} = \mathbf{u}_0$ where $\mathbf{u}_0 = \max(\mathbf{c}, \mathbf{0})$. Thus, if $\mathbf{1}^\top \mathbf{u} \leq 1$ then $\mathbf{u}_0$ is the optimal solution. Otherwise, ν must be different from 0. In this case, from (A.20), any optimal solution must satisfy $\mathbf{1}^\top \mathbf{u} = 1$ and thus, the problem is reduced to part (a).

A.2 ADDITIONAL EXPERIMENTAL RESULTS

In this section, we provide additional experimental results, including the running time for each algorithm.

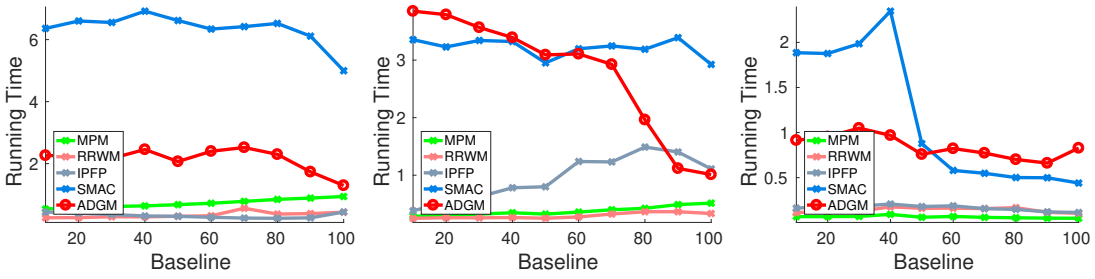
A.2.1 *House and Hotel dataset*

In Section 5.4.1 we presented the results on the House sequence for Pairwise Model B. Results for the Hotel sequence are given in Figure A.1 below. We also report the running time for these experiments in Figure A.2 and Figure A.3.



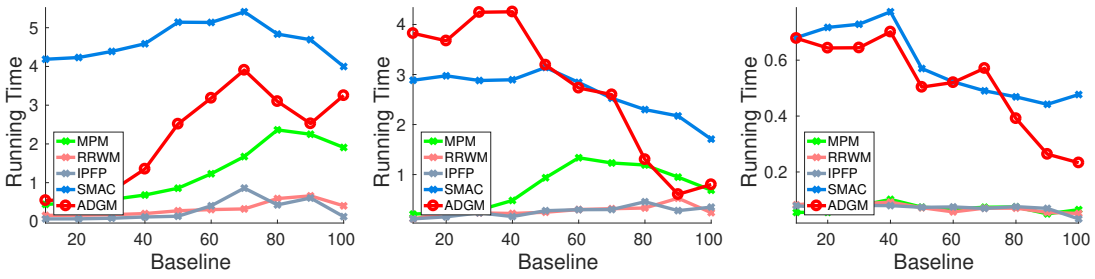
(a) 30 *vs.* 30 points (b) Hotel: 20 *vs.* 30 points (c) Hotel: 10 *vs.* 30 points

FIGURE A.1 Results on the **Hotel** sequence using **Pairwise Model B**.



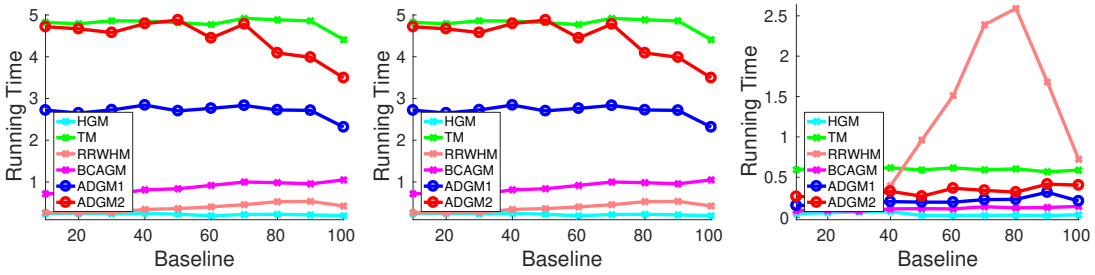
(a) 30 *vs.* 30 points (b) 20 *vs.* 30 points (c) 20 *vs.* 30 points

FIGURE A.2 Running time on the **House** sequence using **Pairwise Model B**.



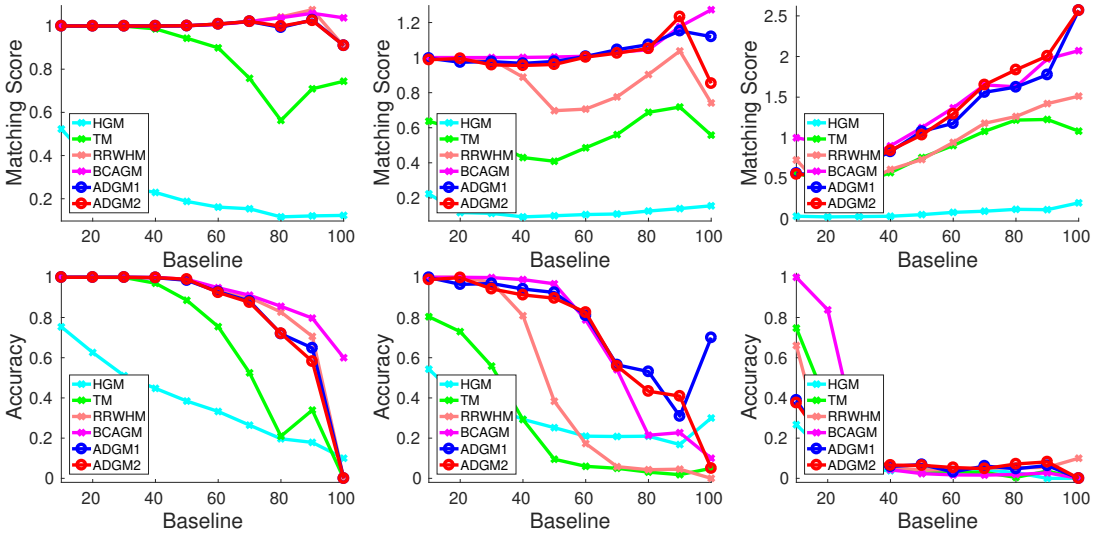
(a) 30 *vs.* 30 points (b) 30 *vs.* 30 points (c) 30 *vs.* 30 points

FIGURE A.3 Running time on the **Hotel** sequence using **Pairwise Model B**.



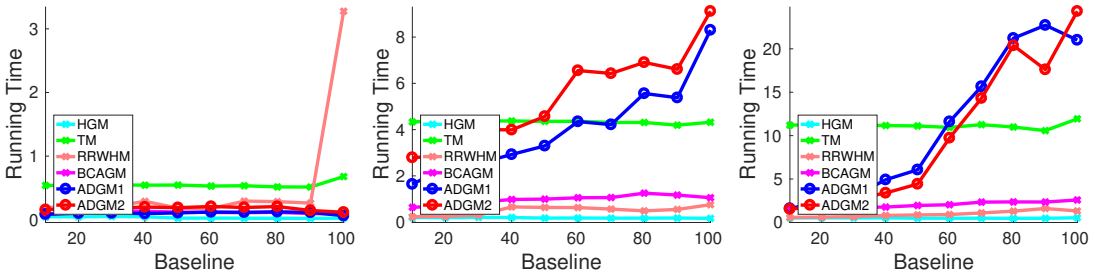
(a) 20 *vs.* 30 points (b) 20 *vs.* 30 points (c) 10 *vs.* 30 points

FIGURE A.4 Running time on the **House** sequence using **Third-order Model**.



(a) 30 *vs.* 30 points (b) 20 *vs.* 30 points (c) 10 *vs.* 30 points

FIGURE A.5 Results on the **Hotel** sequence using **Third-order Model**.

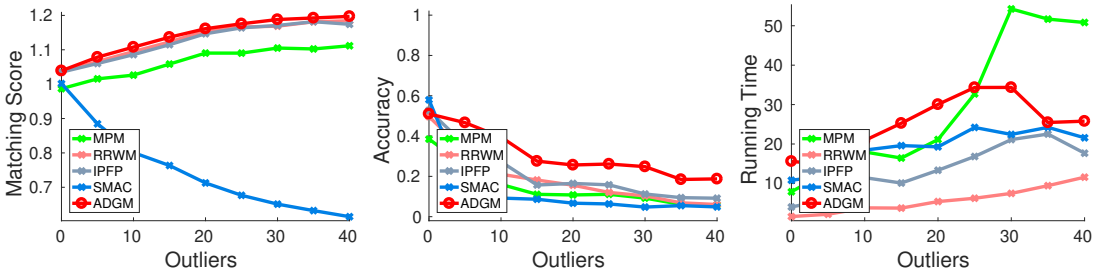


(a) Hotel: 10 pts vs 30 pts (b) Hotel: 20 pts vs 30 pts (c) Hotel: 30 pts vs 30 pts

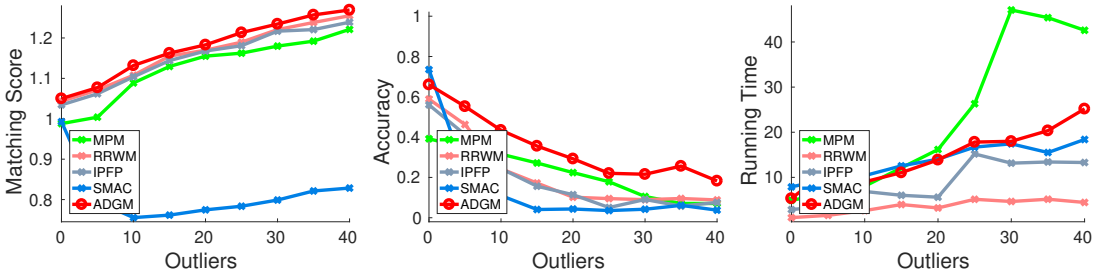
FIGURE A.6 Running time on the **Hotel** sequence using **Third-order Model**.

A.2.2 Cars and Motorbikes

We stated in Section 5.4.2 that using Pairwise Model B (described in Section 5.4.1), the obtained results are unsatisfactory. Indeed, one can observe from Figure A.7 that the obtained matching accuracy is very low, even though **ADGM always achieved the best objective values that are higher than the ground-truth ones**. One can conclude that this pairwise model is not suited for this dataset.



(a) Cars



(b) Motorbikes

FIGURE A.7 Results on the **Cars and Motorbikes** dataset using **Pairwise Model B**, defined in Section 5.4.1. ADGM always achieved the best objective values that are higher than the ground-truth ones. However, the obtained accuracy is still very low. One can conclude that Pairwise Model B is not suited for this dataset.

B

Theoretical Proofs and Additional Details for Chapter 6

B.1 PROOFS OF THEORETICAL RESULTS

B.1.1 Proof of Equation (6.41)

Recall from (6.29) that

$$F(\mathbf{x}^1, \dots, \mathbf{x}^D) = \sum_{\alpha=1}^{D} \sum_{i_1 \dots i_\alpha \in \mathcal{C}} \mathbf{F}_{i_1 \dots i_\alpha} \bigotimes \left\{ \mathbf{x}^1_{i_1}, \dots, \mathbf{x}^\alpha_{i_\alpha} \right\}, \tag{B.1}$$

Clearly, the terms corresponding to any $\alpha < d$ do not involve $\mathbf{x}^d$. Thus, we can rewrite the above as

$$F(\mathbf{x}^1, \dots, \mathbf{x}^D) = \mathrm{cst}(\mathbf{x}^\mathrm{d}) + \sum_{\alpha=d}^{D} \sum_{i_1 \dots i_\alpha \in \mathcal{C}} \mathbf{F}_{i_1 \dots i_\alpha} \bigotimes \left\{ \mathbf{x}^1_{i_1}, \dots, \mathbf{x}^\alpha_{i_\alpha} \right\}. \tag{B.2}$$

We will show that the last double sum can be written as $\sum_{i \in \mathcal{V}} \langle \mathbf{p}^d_i, \mathbf{x}^d_i \rangle$, where $\mathbf{p}^d_i$ is given by (6.41). The idea is to regroup, for each node i, all terms that contain $\mathbf{x}_i$. Indeed, for a given d we have the identity:

$$\sum_{i_1 i_2 \dots i_\alpha \in \mathcal{C}} = \sum_{i_d \in \mathcal{V}} \sum_{i_1 \dots i_{d-1}} \sum_{i_d i_{d+1} \dots i_\alpha \in \mathcal{C}}. \tag{B.3}$$

Therefore, the double sum in (B.2) becomes

$$\sum_{\alpha=d}^{D} \sum_{i_d \in \mathcal{V}} \sum_{i_1 \dots i_{d-1}} \sum_{i_d i_{d+1} \dots i_\alpha \in \mathcal{C}} \mathbf{F}_{i_1 \dots i_\alpha} \bigotimes \left\{ \mathbf{x}^1_{i_1}, \dots, \mathbf{x}^\alpha_{i_\alpha} \right\}. \tag{B.4}$$

Rearranging the first and second sums we obtain

$$\sum_{i_d \in \mathcal{V}} \sum_{\alpha=d}^{D} \sum_{i_1 \dots i_{d-1}} \sum_{i_d i_{d+1} \dots i_\alpha \in \mathcal{C}} \mathbf{F}_{i_1 \dots i_\alpha} \bigotimes \left\{ \mathbf{x}^1_{i_1}, \dots, \mathbf{x}^\alpha_{i_\alpha} \right\}. \tag{B.5}$$

With the change of variable $i \leftarrow i_d$ this becomes

$$\sum_{i\in\mathcal{V}} \sum_{\alpha=d}^{D} \sum_{i_1\dots i_{d-1} i i_{d+1}\dots i_\alpha \in \mathcal{C}} \mathbf{F}_{i_1\dots i_\alpha} \bigotimes \left\{ \mathbf{x}_{i_1}^1, \dots, \mathbf{x}_{i_\alpha}^\alpha \right\}. \tag{B.6}$$

Now by factoring out $\mathbf{x}_i^d$ for each $i \in \mathcal{V}$ the above becomes

$$\sum_{i\in\mathcal{V}} \left(\sum_{\alpha=d}^{D} \sum_{i_1\dots i_{d-1} i i_{d+1}\dots i_\alpha \in \mathcal{C}} \mathbf{F}_{i_1 i_2 \dots i_\alpha} \bigotimes \left\{ \mathbf{x}_{i_1}^1, \dots, \mathbf{x}_{i_{d-1}}^{d-1}, \mathbf{x}_{i_{d+1}}^{d+1}, \dots, \mathbf{x}_{i_\alpha}^\alpha \right\} \right)^\top \mathbf{x}_i^d, \tag{B.7}$$

which is $\sum_{i\in\mathcal{V}} \langle \mathbf{p}_i^d, \mathbf{x}_i^d \rangle$, where $\mathbf{p}_i^d$ is given by (6.41), QED.

B.1.2 *Proof of Equations (6.45)–(6.47)*

See Appendix B.2.2, page 113 on the details of ADMM.

B.1.3 *Proof of Proposition 6.2*

For PGD and FW, the result holds for general continuously differentiable function $E(\cdot)$ and closed convex set $\mathcal{X}$. We refer to [Bertsekas, 1999] (Sections 2.2.2 and 2.3.2) for a proof. Below we give a proof for BCD.

In Proposition 6.1 we have shown that BCD reaches a discrete fixed point $\mathbf{x}^{(k)}$ after a finite number of iterations k. Now, we show that this fixed point is stationary. Define $\Delta_i = \left\{ \mathbf{u} \in \mathbb{R}^{|\mathcal{S}_i|} : \mathbf{u} \geq \mathbf{0}, \mathbf{1}^\top \mathbf{u} = 1 \right\} \ \forall i \in \mathcal{V}$ and let $\mathbf{x}^* = \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$. At the last i^{th} inner iteration (6.7) we have:

$$E(\mathbf{x}_{[1,i-1]}^{(k+1)}, \mathbf{x}_i, \mathbf{x}_{[i+1,n]}^{(k)}) \geq E(\mathbf{x}_{[1,i-1]}^{(k+1)}, \mathbf{x}_i^{(k+1)}, \mathbf{x}_{[i+1,n]}^{(k)}) \tag{B.8}$$

for all $\mathbf{x}_i \in \Delta_i$, which is

$$E(\mathbf{x}_{[1,i-1]}^{*}, \mathbf{x}_i, \mathbf{x}_{[i+1,n]}^{*}) \geq E(\mathbf{x}_{[1,i-1]}^{*}, \mathbf{x}_i^{*}, \mathbf{x}_{[i+1,n]}^{*}) \tag{B.9}$$

for all $\mathbf{x}_i \in \Delta_i$. Define for each i the function

$$E_i^*(\mathbf{x}_i) = E(\mathbf{x}_1^*, \dots, \mathbf{x}_{i-1}^*, \mathbf{x}_i, \mathbf{x}_{i+1}^*, \dots, \mathbf{x}_n^*). \tag{B.10}$$

Obviously $E_i^*(\mathbf{x}_i)$ is continuously differentiable as it is linear. Since $\mathbf{x}_i^*$ is a minimizer of $E_i^*(\mathbf{x}_i)$ over Δ_i, which is closed and convex, according to (6.48) (which is a necessary optimality condition) we have $\nabla E_i^*(\mathbf{x}_i^*)^\top (\mathbf{x}_i - \mathbf{x}_i^*) \geq 0 \ \forall \mathbf{x}_i \in \Delta_i$. Notice that

$$\nabla E(\mathbf{x}^*) = \begin{bmatrix} \frac{\partial E(\mathbf{x}^*)}{\partial \mathbf{x}_1} \\ \vdots \\ \frac{\partial E(\mathbf{x}^*)}{\partial \mathbf{x}_n} \end{bmatrix} = \begin{bmatrix} \nabla E_1^*(\mathbf{x}_1^*) \\ \vdots \\ \nabla E_n^*(\mathbf{x}_n^*) \end{bmatrix}, \tag{B.11}$$

we have

$$\nabla E(\mathbf{x}^*)^\top (\mathbf{x} - \mathbf{x}^*) = \sum_{i=1}^{n} \nabla E_i^*(\mathbf{x}_i^*)^\top (\mathbf{x}_i - \mathbf{x}_i^*). \tag{B.12}$$

Since each term in the last sum is non-negative, we have $\nabla E(\mathbf{x}^*)^\top(\mathbf{x}-\mathbf{x}^*) \geq 0\ \forall \mathbf{x} \in \mathcal{X}$, *i.e.* $\mathbf{x}^*$ is stationary.

B.1.4 *Proof of Proposition 6.3*

By Definition 2, a point $(\mathbf{x}^1, \dots, \mathbf{x}^D, \mathbf{y})$ is a KKT of (6.30) if and only if it has the form $(\mathbf{x}^*, \dots, \mathbf{x}^*, \mathbf{y}^*)$ (where $\mathbf{x}^* \in \mathcal{X}$) and at the same time satisfies

$$\mathbf{x}^{*d} \in \operatorname*{argmin}_{\mathbf{x}^d \in \mathcal{X}^d} \left\{ F(\mathbf{x}^*, \dots, \mathbf{x}^*, \mathbf{x}^d, \mathbf{x}^*, \dots, \mathbf{x}^*) + \mathbf{y}^{*\top} \mathbf{A}^d \mathbf{x}^d \right\} \tag{B.13}$$

for all d, which is *equivalent* to

$$\left(\frac{\partial F}{\partial \mathbf{x}^d}(\mathbf{x}^*, \dots, \mathbf{x}^*) + \mathbf{A}^{d\top} \mathbf{y}^* \right)^\top (\mathbf{x}^d - \mathbf{x}^*) \geq 0 \quad \forall \mathbf{x}^d \in \mathcal{X}^d, \forall d. \tag{B.14}$$

The equivalence ("$\Leftrightarrow$") follows from the fact that the objective function (with respect to $\mathbf{x}^d$) in (B.13) is convex. This is a well-known result in convex analysis, which we refer to Bertsekas, Dimitri P., Angelia Nedi, and Asuman E. Ozdaglar. *Convex analysis and optimization.*" (2003) (Proposition 4.7.2) for a proof. Note that from the necessary optimality condition (6.48) we can only have the "$\Rightarrow$" direction.

We need to prove that the sequence $\{(\mathbf{x}^{1^{(k)}}, \dots, \mathbf{x}^{D^{(k)}}, \mathbf{y}^{(k)})\}$ generated by ADMM satisfies the above conditions (under the assumption that the residual $r^{(k)}$ converges to 0).

Let $(\mathbf{x}^{*1}, \mathbf{x}^{*2}, \dots, \mathbf{x}^{*D}, \mathbf{y}^*)$ be a limit point of $\{(\mathbf{x}^{1^{(k)}}, \dots, \mathbf{x}^{D^{(k)}}, \mathbf{y}^{(k)})\}$ (thus $\mathbf{x}^{*d} \in \mathcal{X}^d\ \forall d$ since $(\mathcal{X}^d)_{1 \leq d \leq D}$ are closed), and define a subsequence that converges to this limit point by $\{(\mathbf{x}^{1^{(l)}}, \dots, \mathbf{x}^{D^{(l)}}, \mathbf{y}^{(l)})\}$, $l \in \mathcal{L} \subset \mathbb{N}$ where $\mathcal{L}$ denotes the set of indices of this subsequence. We have

$$\lim_{\substack{l \to +\infty \\ l \in \mathcal{L}}} (\mathbf{x}^{1^{(l)}}, \dots, \mathbf{x}^{D^{(l)}}, \mathbf{y}^{(l)}) = (\mathbf{x}^{*1}, \mathbf{x}^{*2}, \dots, \mathbf{x}^{*D}, \mathbf{y}^*). \tag{B.15}$$

Since the residual $r^{(k)}$ (6.39) converges to 0, we have

$$\lim_{\substack{l \to +\infty \\ l \in \mathcal{L}}} \left(\sum_{d=1}^{D} \mathbf{A}^d \mathbf{x}^{d^{(l)}} \right) = \mathbf{0}, \tag{B.16}$$

$$\lim_{\substack{l \to +\infty \\ l \in \mathcal{L}}} \left(\mathbf{x}^{d^{(l+1)}} - \mathbf{x}^{d^{(l)}} \right) = \mathbf{0} \quad \forall d. \tag{B.17}$$

On the one hand, combining (B.15) and (B.17) we get

$$\lim_{\substack{l \to +\infty \\ l \in \mathcal{L}}} (\mathbf{x}^{1^{(l+1)}}, \dots, \mathbf{x}^{D^{(l+1)}}, \mathbf{y}^{(l+1)}) = (\mathbf{x}^{*1}, \mathbf{x}^{*2}, \dots, \mathbf{x}^{*D}, \mathbf{y}^*). \tag{B.18}$$

(Note that the above is different from (B.15) because $l+1$ might not belong to $\mathcal{L}$.) On the other hand, combining (B.15) and (B.16) we get

$$\sum_{d=1}^{D} \mathbf{A}^d \mathbf{x}^{*d} = \mathbf{0}, \tag{B.19}$$

which is, according to (6.31), equivalent to

$$\mathbf{x}^{*1} = \mathbf{x}^{*2} = \cdots = \mathbf{x}^{*D}. \tag{B.20}$$

Let $\mathbf{x}^* \in \mathcal{X}$ denote the value of these vectors. From (B.15) and (B.18) we have

$$\lim_{\substack{l \to +\infty \\ l \in \mathcal{L}}} \mathbf{x}^{d^{(l)}} = \lim_{\substack{l \to +\infty \\ l \in \mathcal{L}}} \mathbf{x}^{d^{(l+1)}} = \mathbf{x}^* \quad \forall d, \tag{B.21}$$

$$\lim_{\substack{l \to +\infty \\ l \in \mathcal{L}}} \mathbf{y}^{(l)} = \lim_{\substack{l \to +\infty \\ l \in \mathcal{L}}} \mathbf{y}^{(l+1)} = \mathbf{y}^*. \tag{B.22}$$

It only remains to prove that $(\mathbf{x}^*, \dots, \mathbf{x}^*, \mathbf{y}^*)$ satisfies (B.14). Let us denote for convenience

$$\mathbf{z}_d^{(k)} = (\mathbf{x}^{[1,d]^{(k+1)}}, \mathbf{x}^{[d+1,D]^{(k)}}) \quad \forall d. \tag{B.23}$$

According to (6.48), the $\mathbf{x}$ update (6.37) implies

$$\left(\frac{\partial L_\rho}{\partial \mathbf{x}^d}(\mathbf{z}_d^{(k)}, \mathbf{y}^{(k)})\right)^\top \left(\mathbf{x}^d - \mathbf{x}^{d^{(k+1)}}\right) \geq 0 \quad \forall \mathbf{x}^d \in \mathcal{X}^d, \forall d, \forall k. \tag{B.24}$$

Since L_ρ (6.36) is continuously differentiable, applying (B.21) and (B.22) we obtain

$$\lim_{l \to +\infty l \in \mathcal{L}} \frac{\partial L_\rho}{\partial \mathbf{x}^d}(\mathbf{z}_d^{(l)}, \mathbf{y}^{(l)}) = \frac{\partial L_\rho}{\partial \mathbf{x}^d}(\mathbf{x}^*, \dots, \mathbf{x}^*, \mathbf{y}^*) \quad \forall d. \tag{B.25}$$

Let $k = l$ in (B.24) and take the limit of that inequality, taking into account (B.21) and (B.25), we get

$$\left(\frac{\partial L_\rho}{\partial \mathbf{x}^d}(\mathbf{x}^*, \dots, \mathbf{x}^*, \mathbf{y}^*)\right)^\top (\mathbf{x}^d - \mathbf{x}^*) \geq 0 \quad \forall \mathbf{x}^d \in \mathcal{X}^d, \forall d. \tag{B.26}$$

From the definition of L_ρ (6.36) we have

$$\begin{aligned} \frac{\partial L_\rho}{\partial \mathbf{x}^d}(\mathbf{x}^*, \dots, \mathbf{x}^*, \mathbf{y}^*) &= \frac{\partial F}{\partial \mathbf{x}^d}(\mathbf{x}^*, \dots, \mathbf{x}^*) + \mathbf{A}^{d\top}\mathbf{y}^* + \rho \mathbf{A}^{d\top}\left(\sum_{d=1}^{D} \mathbf{A}^d \mathbf{x}^*\right) \\ &= \frac{\partial F}{\partial \mathbf{x}^d}(\mathbf{x}^*, \dots, \mathbf{x}^*) + \mathbf{A}^{d\top}\mathbf{y}^*. \end{aligned} \tag{B.27}$$

Note that the last equality follows from (6.31). Plugging the above into the last inequality we obtain

$$\left(\frac{\partial F}{\partial \mathbf{x}^d}(\mathbf{x}^*, \dots, \mathbf{x}^*) + \mathbf{A}^{d\top}\mathbf{y}^*\right)^\top (\mathbf{x}^d - \mathbf{x}^*) \geq 0 \quad \forall \mathbf{x}^d \in \mathcal{X}^d, \forall d, \tag{B.28}$$

which is exactly (B.14), and this completes the proof.

B.1.5 *Proof of Proposition 6.4*

Let $(\mathbf{x}^*, \dots, \mathbf{x}^*, \mathbf{y}^*)$ be a KKT point of (6.30). We have seen in the previous proof that

$$\left(\frac{\partial F}{\partial \mathbf{x}^d}(\mathbf{x}^*, \dots, \mathbf{x}^*) + \mathbf{A}^{d\top}\mathbf{y}^*\right)^\top (\mathbf{x}^d - \mathbf{x}^*) \geq 0 \quad \forall \mathbf{x}^d \in \mathcal{X}^d, \forall d. \tag{B.29}$$

According to (6.40):

$$\frac{\partial F}{\partial \mathbf{x}^d}(\mathbf{x}^1, \dots, \mathbf{x}^D) = \mathbf{p}^d, \tag{B.30}$$

where $\mathbf{p}^d$ is defined by (6.41). Now let $\mathbf{p}^{*d}$ be the value of $\mathbf{p}^d$ where $(\mathbf{x}^1, \dots, \mathbf{x}^D)$ is replaced by $(\mathbf{x}^*, \dots, \mathbf{x}^*)$, *i.e.* $\mathbf{p}^{*d} = (\mathbf{p}_1^{*d}, \dots, \mathbf{p}_n^{*d})$ where

$$\mathbf{p}_i^{*d} = \sum_{\alpha=d}^{D} \left(\sum_{i_1 \dots i_{d-1} i i_{d+1} \dots i_\alpha \in \mathcal{C}} \mathbf{F}_{i_1 i_2 \dots i_\alpha} \bigotimes \left\{ \mathbf{x}_{i_1}^*, \dots, \mathbf{x}_{i_{d-1}}^*, \mathbf{x}_{i_{d+1}}^*, \dots, \mathbf{x}_{i_\alpha}^* \right\} \right) \forall i \in \mathcal{V}. \tag{B.31}$$

Notice that $\frac{\partial F}{\partial \mathbf{x}^d}(\mathbf{x}^*, \dots, \mathbf{x}^*) = \mathbf{p}^{*d}$, (B.29) becomes

$$\left(\mathbf{p}^{*d} + \mathbf{A}^{d\top}\mathbf{y}^*\right)^\top (\mathbf{x}^d - \mathbf{x}^*) \geq 0 \quad \forall \mathbf{x}^d \in \mathcal{X}^d, \forall d. \tag{B.32}$$

According to (6.32) we have $\mathcal{X} \subseteq \mathcal{X}^d$ and therefore the above inequality implies

$$\left(\mathbf{p}^{*d} + \mathbf{A}^{d\top}\mathbf{y}^*\right)^\top (\mathbf{x} - \mathbf{x}^*) \geq 0 \quad \forall \mathbf{x} \in \mathcal{X}, \forall d. \tag{B.33}$$

Summing this inequality for all d we get

$$\left(\sum_{d=1}^{D} \mathbf{p}^{*d}\right)^\top (\mathbf{x} - \mathbf{x}^*) + \mathbf{y}^{*\top} \left(\sum_{d=1}^{D} \mathbf{A}^d\right) (\mathbf{x} - \mathbf{x}^*) \geq 0 \quad \forall \mathbf{x} \in \mathcal{X}. \tag{B.34}$$

Yet, according to (6.31) we have $\sum_{d=1}^{D} \mathbf{A}^d \mathbf{x} = \sum_{d=1}^{D} \mathbf{A}^d \mathbf{x}^* = \mathbf{0}$. Therefore, the second term in the above inequality is 0, yielding

$$\left(\sum_{d=1}^{D} \mathbf{p}^{*d}\right)^\top (\mathbf{x} - \mathbf{x}^*) \geq 0 \quad \forall \mathbf{x} \in \mathcal{X}. \tag{B.35}$$

Now if we can prove that

$$\sum_{d=1}^{D} \mathbf{p}^{*d} = \nabla E(\mathbf{x}^*), \tag{B.36}$$

then we have $\nabla E(\mathbf{x}^*)^\top (\mathbf{x} - \mathbf{x}^*) \geq 0 \ \forall \mathbf{x} \in \mathcal{X}$ and thus according to Definition 1, $\mathbf{x}^*$ is a stationary point of (RLX).

Let us now prove (B.36). Indeed, we can rewrite (B.31) as

$$\mathbf{p}_i^{*d} = \sum_{\alpha=d}^{D} \left(\sum_{\substack{C \in \mathcal{C} \\ C = (i_1 \dots i_{d-1} i i_{d+1} \dots i_\alpha)}} \mathbf{F}_C \bigotimes \left\{ \mathbf{x}_j^* \right\}_{j \in C \setminus i} \right) \forall i \in \mathcal{V}. \tag{B.37}$$

Therefore,

$$\sum_{d=1}^{D} \mathbf{p}_i^{*d} = \sum_{d=1}^{D} \sum_{\alpha=d}^{D} \sum_{\substack{C \in \mathcal{C} \\ C=(i_1 \dots i_{d-1} i i_{d+1} \dots i_\alpha)}} \mathbf{F}_C \bigotimes \left\{\mathbf{x}_j^*\right\}_{j \in C \setminus i} \forall i \in \mathcal{V}. \tag{B.38}$$

Let's take a closer look at this triple sum. The double sum

$$\sum_{\alpha=d}^{D} \sum_{\substack{C \in \mathcal{C} \\ C=(i_1 \dots i_{d-1} i i_{d+1} \dots i_\alpha)}}$$

basically means *iterating through all cliques whose sizes are* $\geq d$ *and whose* d^{th} *node is* i. Obviously the condition "sizes $\geq d$" is redundant here, thus the above means *iterating through all cliques whose* d^{th} *node is* i. Combined with $\sum_{d=1}^{D}$, the above triple sum means *for each size* d*, iterating through all cliques whose* d^{th} *node is* i, which is clearly equivalent to *iterating through all cliques that contain* i. Therefore, (B.38) can be rewritten more compactly as

$$\sum_{d=1}^{D} \mathbf{p}_i^{*d} = \sum_{C \in \mathcal{C}(i)} \mathbf{F}_C \bigotimes \left\{\mathbf{x}_j^*\right\}_{j \in C \setminus i} \quad \forall i \in \mathcal{V}, \tag{B.39}$$

where $\mathcal{C}(i)$ is the set of cliques that contain the node i. Recall from (6.10) that the last expression is actually $\frac{\partial E(\mathbf{x}^*)}{\partial \mathbf{x}^i}$, *i.e.*

$$\sum_{d=1}^{D} \mathbf{p}_i^{*d} = \frac{\partial E(\mathbf{x}^*)}{\partial \mathbf{x}^i} \quad \forall i \in \mathcal{V}, \tag{B.40}$$

or equivalently

$$\sum_{d=1}^{D} \mathbf{p}^{*d} = \nabla E(\mathbf{x}^*), \tag{B.41}$$

which is (B.36), and this completes the proof.

B.2 MORE DETAILS ON THE IMPLEMENTED METHODS

B.2.1 *Convex QP relaxation*

This relaxation was presented in [Ravikumar and Lafferty, 2006] for pairwise MRFs (6.20). Define:

$$d_i(s) = \sum_{j \in \mathcal{N}(i)} \sum_{t \in \mathcal{S}_j} \frac{1}{2} \left| f_{ij}(s,t) \right|. \tag{B.42}$$

Denote $\mathbf{d}_i = (d_i(s))_{s \in \mathcal{S}_i}$ and $\mathbf{D}_i = \mathrm{diag}(\mathbf{d}_i)$, the diagonal matrix composed by $\mathbf{d}_i$. The convex QP relaxation energy is given by

$$E_{\mathrm{cqp}}(\mathbf{x}) = E_{\mathrm{pairwise}}(\mathbf{x}) - \sum_{i \in \mathcal{V}} \mathbf{d}_i^\top \mathbf{x}_i + \sum_{i \in \mathcal{V}} \mathbf{x}_i^\top \mathbf{D}_i \mathbf{x}_i. \tag{B.43}$$

This convex energy can be minimized using different methods. Here we propose to solve it using Frank-Wolfe algorithm, which has the guarantee to reach the global optimum.

Similarly to the previous nonconvex Frank-Wolfe algorithm, the update step (6.14) can be solved using Lemma 6.1, and the line-search has closed-form solutions:

$$\begin{aligned} E_{\text{cqp}}(\mathbf{x}+\alpha\mathbf{r}) &= E_{\text{pairwise}}(\mathbf{x}+\alpha\mathbf{r}) - \sum_{i\in\mathcal{V}} \mathbf{d}_i^\top(\mathbf{x}_i+\alpha\mathbf{r}_i) + \sum_{i\in\mathcal{V}} (\mathbf{x}_i+\alpha\mathbf{r}_i)^\top \mathbf{D}_i(\mathbf{x}_i+\alpha\mathbf{r}_i) \\ &= A'\alpha^2 + B'\alpha + C', \end{aligned} \tag{B.44}$$

where

$$A' = A + \sum_{i\in\mathcal{V}} \mathbf{r}_i^\top \mathbf{D}_i \mathbf{r}_i \tag{B.45}$$

$$B' = B + \sum_{i\in\mathcal{V}} \left(-\mathbf{d}_i^\top \mathbf{r}_i + \mathbf{r}_i^\top \mathbf{D}_i \mathbf{x}_i + \mathbf{x}_i^\top \mathbf{D}_i \mathbf{r}_i\right) \tag{B.46}$$

$$C' = C + \sum_{i\in\mathcal{V}} \left(-\mathbf{d}_i^\top \mathbf{x}_i + \mathbf{x}_i^\top \mathbf{D}_i \mathbf{x}_i\right). \tag{B.47}$$

B.2.2 ADMM

In this section, we give more details on the instantiation of ADMM into different decompositions. As we have seen in Section 6.4.2, there is an infinite number of such decompositions. Some examples include:

$$\text{(cyclic)} \qquad \mathbf{x}^{d-1} = \mathbf{x}^d, \quad d = 2, \dots, D, \tag{B.48}$$

$$\text{(star)} \qquad \mathbf{x}^1 = \mathbf{x}^d, \quad d = 2, \dots, D, \tag{B.49}$$

$$\text{(symmetric)} \qquad \mathbf{x}^d = (\mathbf{x}^1 + \cdots + \mathbf{x}^D)/D \quad \forall d. \tag{B.50}$$

Let us consider for example the *cyclic* decomposition. We obtain the following problem, equivalent to (RLX):

$$\begin{aligned} \min \quad & F(\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^D) \\ \text{s.t.} \quad & \mathbf{x}^{d-1} = \mathbf{x}^d, \quad d = 2, \dots, D, \\ & \mathbf{x}^d \in \mathcal{X}^d, \qquad d = 1, \dots, D, \end{aligned} \tag{B.51}$$

where $\mathcal{X}^1, \dots, \mathcal{X}^D$ are closed convex sets satisfying $\mathcal{X}^1 \cap \mathcal{X}^2 \cap \cdots \cap \mathcal{X}^D = \mathcal{X}$, and F is defined by (6.29).

The augmented Lagrangian of this problem is:

$$L_\rho(\mathbf{x}^1, \dots, \mathbf{x}^D, \mathbf{y}) = F(\mathbf{x}^1, \dots, \mathbf{x}^D) + \sum_{d=2}^{D} \langle \mathbf{y}^d, \mathbf{x}^{d-1} - \mathbf{x}^d \rangle + \frac{\rho}{2} \sum_{d=2}^{D} \left\| \mathbf{x}^{d-1} - \mathbf{x}^d \right\|_2^2, \tag{B.52}$$

where $\mathbf{y} = (\mathbf{y}^2, \dots, \mathbf{y}^D)$. The $\mathbf{y}$ update (6.38) becomes

$$\mathbf{y}^{d(k+1)} = \mathbf{y}^{d(k)} + \rho \left(\mathbf{x}^{d-1(k+1)} - \mathbf{x}^{d(k+1)} \right). \tag{B.53}$$

Consider the $\mathbf{x}$ update (6.37). Plugging (6.40) into (B.52), expanding and regrouping,

we obtain that $L_\rho(\mathbf{x}^1, \dots, \mathbf{x}^D, \mathbf{y})$ is equal to each of the following expressions:

$$\frac{\rho}{2}\|\mathbf{x}^1\|_2^2 - \langle \mathbf{x}^1, \rho\mathbf{x}^2 - \mathbf{y}^2 - \mathbf{p}^1\rangle + \mathrm{cst}(\mathbf{x}^1), \tag{B.54}$$

$$\rho\|\mathbf{x}^d\|_2^2 - \langle \mathbf{x}^d, \rho\mathbf{x}^{d-1} + \rho\mathbf{x}^{d+1} + \mathbf{y}^d - \mathbf{y}^{d+1} - \mathbf{p}^d\rangle + \mathrm{cst}(\mathbf{x}^d) \qquad (2 \leq d \leq D-1), \tag{B.55}$$

$$\frac{\rho}{2}\|\mathbf{x}^D\|_2^2 - \langle \mathbf{x}^D, \rho\mathbf{x}^{D-1} + \mathbf{y}^D - \mathbf{p}^D\rangle + \mathrm{cst}(\mathbf{x}^D). \tag{B.56}$$

From this, it is straightforward to see that the $\mathbf{x}$ update (6.37) is reduced to (6.44) where $(\mathbf{c}_d)_{1\leq d\leq D}$ are defined by (6.45), (6.46) and (6.47).

It is straightforward to obtain similar results for the other decompositions.

B.3 DETAILED EXPERIMENTAL RESULTS

In Section 6.6 we presented the results averaged over all problem instances. We provide below the details for every single instance.

TABLE B.1 Inpainting N4.

inpainting-n4		FastPD	α-Exp	TRBP	ADDD	MPLP	MPLP-C	TRWS	BUNDLE
triplepoint4-plain-ring-inverse	value	424.90	**424.12**	475.95	482.23	508.94	453.17	496.37	425.90
	bound	205.21	-Inf	-Inf	402.83	339.29	411.48	411.59	**411.87**
	runtime	0.03	0.02	33.04	28.59	1.75	3615.97	2.15	44.41
triplepoint4-plain-ring	value	**484.59**	**484.59**	**484.59**	**484.59**	485.38	**484.59**	**484.59**	**484.59**
	bound	384.57	-Inf	-Inf	**484.59**	484.58	484.59	**484.59**	484.59
	runtime	0.03	0.02	13.85	3.15	108.89	118.43	0.59	27.96
mean energy		454.75	454.35	480.27	483.41	497.16	468.88	467.70	455.25
mean bound		294.89	-Inf	-Inf	443.71	411.94	448.03	448.09	448.23
mean runtime		0.03	0.02	23.45	15.87	55.32	1867.20	1.37	36.18
best value		50.00	100.00	50.00	50.00	0.00	50.00	50.00	50.00
best bound		0.00	0.00	0.00	50.00	0.00	0.00	50.00	50.00
verified opt		0.00	0.00	0.00	50.00	0.00	0.00	50.00	0.00

TABLE B.2 Inpainting N4.

inpainting-n4		CQP	ADMM	BCD	FW	PGD
triplepoint4-plain-ring-inverse	value	2256.45	**424.12**	443.18	443.18	444.75
	bound	-Inf	-Inf	-Inf	-Inf	-Inf
	runtime	2.60	7.69	0.11	1.05	0.77
triplepoint4-plain-ring	value	542.57	**484.59**	528.57	533.29	534.86
	bound	-Inf	-Inf	-Inf	-Inf	-Inf
	runtime	1.24	12.00	0.11	1.15	0.85
mean energy		490.09	454.35	485.88	488.23	489.80
mean bound		-Inf	-Inf	-Inf	-Inf	-Inf
mean runtime		1.92	9.84	0.11	1.10	0.81
best value		0.00	100.00	0.00	0.00	0.00
best bound		0.00	0.00	0.00	0.00	0.00
verified opt		0.00	0.00	0.00	0.00	0.00

TABLE B.3 Inpainting N8.

inpainting-n8		α-Exp	FastPD	TRBP	ADDD	MPLP	MPLP-C	BUNDLE	TRWS
triplepoint4-plain-ring-inverse	value	434.84	434.84	496.40	714.42	442.42	463.88	435.32	504.97
	bound	-Inf	0.00	-Inf	406.71	412.37	413.49	**415.83**	413.20
	runtime	0.90	0.19	97.95	57.01	1107.98	3660.44	112.91	16.09
triplepoint4-plain-ring	value	**495.20**	**495.20**	**495.20**	495.85	495.52	**495.20**	**495.20**	**495.20**
	bound	-Inf	272.56	-Inf	495.18	494.72	**495.20**	495.04	494.71
	runtime	0.67	0.11	30.04	14.56	581.96	884.34	110.56	16.37
mean energy		465.02	465.02	494.02	605.14	468.83	469.78	465.26	466.80
mean bound		-Inf	136.28	-Inf	450.95	453.55	454.35	455.43	453.96
mean runtime		0.78	0.15	64.00	35.78	844.97	2272.39	111.74	16.23

inpainting-n8		α-Exp	FastPD	TRBP	ADDD	MPLP	MPLP-C	BUNDLE	TRWS
best value		50.00	50.00	50.00	0.00	0.00	50.00	50.00	50.00
best bound		0.00	0.00	0.00	0.00	0.00	50.00	50.00	0.00
verified opt		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

TABLE B.4 Inpainting N8.

inpainting-n8		CQP	ADMM	BCD	FW	PGD
triplepoint4-plain-ring-inverse	value	1819.57	**434.32**	438.95	446.19	446.19
	bound	-Inf	-Inf	-Inf	-Inf	-Inf
	runtime	20.78	38.71	0.31	6.33	6.55
triplepoint4-plain-ring	value	538.25	**495.20**	524.94	533.45	533.45
	bound	-Inf	-Inf	-Inf	-Inf	-Inf
	runtime	2.46	42.57	0.28	5.55	3.83
mean energy		489.82	464.76	481.95	489.82	489.82
mean bound		-Inf	-Inf	-Inf	-Inf	-Inf
mean runtime		11.62	40.64	0.29	5.94	5.19
best value		0.00	100.00	0.00	0.00	0.00
best bound		0.00	0.00	0.00	0.00	0.00
verified opt		0.00	0.00	0.00	0.00	0.00

TABLE B.5 Feature matching.

matching		TRBP	ADDD	MPLP	MPLP-C	BUNDLE	TRWS
matching0	value	60000000075.71	200000000047.27	90000000059.69	**19.36**	58.64	61.05
	bound	-Inf	11.56	10.96	**19.36**	11.27	11.02
	runtime	0.00	2.45	0.22	8.02	1.09	0.04
matching1	value	170000000090.50	70000000031.36	50000000030.34	**23.58**	10000000021.89	102.20
	bound	-Inf	20.13	18.47	**23.58**	17.48	18.52
	runtime	0.00	3.82	0.52	4.52	2.70	0.04
matching2	value	110000000096.00	20000000026.59	30000000025.18	**26.08**	20000000043.93	51.59
	bound	-Inf	22.97	21.07	**26.08**	19.87	21.18
	runtime	0.00	4.12	0.94	8.25	3.56	0.12
matching3	value	80000000066.03	130000000051.70	90000000051.81	**15.86**	10000000042.82	41.92
	bound	-Inf	10.72	10.15	**15.86**	9.25	10.14
	runtime	0.00	2.25	0.21	3.36	1.96	0.01
mean energy		97500000064.52	105000000039.23	65000000041.76	21.22	10000000041.82	63.52
mean bound		-Inf	16.35	15.16	21.22	14.47	15.22
mean runtime		0.00	3.16	0.47	6.04	2.33	0.05
best value		0.00	0.00	0.00	100.00	0.00	0.00
best bound		0.00	0.00	0.00	100.00	0.00	0.00
verified opt		0.00	0.00	0.00	100.00	0.00	0.00

TABLE B.6 Feature matching.

matching		BCD	FW	PGD	CQP	ADMM
matching0	value	43.61	56.10	49.45	118.90	42.09
	bound	-Inf	-Inf	-Inf	-Inf	-Inf
	runtime	0.00	0.19	8.08	0.06	0.02
matching1	value	118.00	77.31	79.01	138.99	107.31
	bound	-Inf	-Inf	-Inf	-Inf	-Inf
	runtime	0.00	23.66	21.36	0.10	0.94
matching2	value	139.74	89.46	62.40	156.46	107.41
	bound	-Inf	-Inf	-Inf	-Inf	-Inf
	runtime	0.00	55.74	19.28	0.08	0.26
matching3	value	38.09	43.98	43.21	93.67	43.69
	bound	-Inf	-Inf	-Inf	-Inf	-Inf
	runtime	0.00	0.81	4.11	0.07	0.02
mean energy		84.86	66.71	58.52	127.01	75.12
mean bound		-Inf	-Inf	-Inf	-Inf	-Inf
mean runtime		0.00	20.10	13.21	0.08	0.31
best value		0.00	0.00	0.00	0.00	0.00
best bound		0.00	0.00	0.00	0.00	0.00
verified opt		0.00	0.00	0.00	0.00	0.00

TABLE B.7 Pairwise stereo.

mrf-stereo		FastPD	α-Exp	TRBP	ADDD	MPLP	MPLP-C	TRWS
ted-gm	value	1344017.00	**1343176.00**	1460166.00	NaN	NaN	NaN	1346202.00
	bound	395613.00	-Inf	-Inf	NaN	NaN	NaN	**1337092.22**
	runtime	14.94	29.75	3616.74	NaN	NaN	NaN	391.34

mrf-stereo		FastPD	α-Exp	TRBP	ADDD	MPLP	MPLP-C	TRWS
tsu-gm	value	370825.00	370255.00	411157.00	455874.00	369304.00	369865.00	369279.00
	bound	31900.00	-Inf	-Inf	299780.16	367001.47	366988.29	369217.58
	runtime	1.72	3.64	1985.50	1066.79	4781.02	4212.26	393.76
ven-gm	value	3127923.00	3138157.00	3122190.00	NaN	NaN	NaN	**3048404.00**
	bound	475665.00	-Inf	-Inf	NaN	NaN	NaN	**3047929.95**
	runtime	4.76	10.87	2030.13	NaN	NaN	NaN	478.49
mean energy		1614255.00	1617196.00	1664504.33	NaN	NaN	NaN	1587596.67
mean bound		301059.33	-Inf	-Inf	NaN	NaN	NaN	1584746.58
mean runtime		7.14	14.75	2544.12	NaN	NaN	NaN	421.20
best value		0.00	33.33	0.00	0.00	0.00	0.00	33.33
best bound		0.00	0.00	0.00	0.00	0.00	0.00	66.67
verified opt		0.00	0.00	0.00	0.00	0.00	0.00	0.00

TABLE B.8 Pairwise stereo.

mrf-stereo		CQP	ADMM	BCD	FW	PGD	BUNDLE
ted-gm	value	4195611.00	1373030.00	3436281.00	3020579.00	2694493.00	1563172.00
	bound	-Inf	-Inf	-Inf	-Inf	-Inf	1334223.01
	runtime	3602.97	3628.80	15.64	1740.10	2109.65	3530.00
tsu-gm	value	3621062.00	375954.00	2722934.00	2352499.00	2114223.00	**369218.00**
	bound	-Inf	-Inf	-Inf	-Inf	-Inf	**369218.00**
	runtime	3600.79	807.70	5.33	622.64	120.38	670.81
ven-gm	value	26408665.00	3123334.00	14907352.00	13114176.00	10818561.00	3061733.00
	bound	-Inf	-Inf	-Inf	-Inf	-Inf	3047785.37
	runtime	3602.28	2696.49	11.48	3604.63	2298.42	1917.58
mean energy		11408446.00	1624106.00	7022189.00	6162418.00	5209092.33	1664707.67
mean bound		-Inf	-Inf	-Inf	-Inf	-Inf	1583742.13
mean runtime		3602.01	2377.66	10.82	1989.12	1509.49	2039.47
best value		0.00	0.00	0.00	0.00	0.00	33.33
best bound		0.00	0.00	0.00	0.00	0.00	33.33
verified opt		0.00	0.00	0.00	0.00	0.00	0.00

TABLE B.9 Segmentation.

inclusion		α-Fusion	TRBP	ADDD	MPLP	MPLP-C	BUNDLE	SRMP	ADMM
modelH-1-0.8-0.2	value	1595.06	1416.07	2416.58	3416.08	5415.89	5427.91	**1415.94**	**1415.94**
	bound	-Inf	-Inf	1415.71	1415.70	1415.71	1406.09	**1415.94**	-Inf
	runtime	0.06	21.93	10.52	12.17	3843.79	99.77	0.11	106.70
modelH-10-0.8-0.2	value	1590.97	1416.80	3415.92	5415.13	4415.43	5422.47	**1416.10**	1416.24
	bound	-Inf	-Inf	1415.68	1415.62	1415.70	1404.47	**1416.10**	-Inf
	runtime	0.05	22.66	1.20	12.03	3797.40	91.16	0.13	85.46
modelH-2-0.8-0.2	value	1603.85	1423.42	4423.49	6422.84	3423.03	5436.16	**1422.89**	**1422.89**
	bound	-Inf	-Inf	1422.79	1422.78	1422.79	1411.56	**1422.89**	-Inf
	runtime	0.05	21.34	10.00	6.67	4051.20	101.83	0.11	113.24
modelH-3-0.8-0.2	value	1596.11	**1381.14**	**1381.14**	**1381.14**	**1381.14**	4389.78	**1381.14**	1381.19
	bound	-Inf	-Inf	**1381.14**	1381.14	1381.14	1371.29	**1381.14**	-Inf
	runtime	0.06	8.02	4.50	7.79	8.84	112.52	0.11	63.51
modelH-4-0.8-0.2	value	1595.12	1427.56	5427.63	5426.48	3427.27	2432.97	**1427.17**	**1427.17**
	bound	-Inf	-Inf	1426.58	1426.56	1426.58	1416.80	**1427.17**	-Inf
	runtime	0.04	21.18	9.40	8.29	3892.65	116.38	0.13	125.01
modelH-5-0.8-0.2	value	1566.58	3383.89	6383.61	4383.52	6382.77	4390.47	**1383.69**	1383.77
	bound	-Inf	-Inf	1383.25	1383.23	1383.30	1371.94	**1383.69**	-Inf
	runtime	0.04	21.05	8.45	5.44	3902.54	112.86	0.18	99.08
modelH-6-0.8-0.2	value	1588.33	2402.30	2402.17	2402.60	5401.70	3406.27	**1402.34**	1402.60
	bound	-Inf	-Inf	1402.01	1401.77	1402.01	1393.05	**1402.34**	-Inf
	runtime	0.03	20.80	2.69	22.61	3778.21	101.74	0.11	126.40
modelH-7-0.8-0.2	value	1583.36	1403.61	3403.70	5402.97	5403.24	6418.08	**1403.25**	1403.69
	bound	-Inf	-Inf	1403.08	1403.07	1403.08	1391.87	**1403.25**	-Inf
	runtime	0.04	20.80	2.50	11.98	4124.95	103.95	0.15	94.36
modelH-8-0.8-0.2	value	1574.64	3368.65	3368.65	3368.66	1368.55	**1368.33**	**1368.33**	**1368.33**
	bound	-Inf	-Inf	1368.29	1368.29	1368.33	1368.23	**1368.33**	-Inf
	runtime	0.05	20.66	11.21	5.09	3740.80	92.39	0.15	86.69
modelH-9-0.8-0.2	value	1577.25	1385.00	1385.23	2385.04	3385.06	**1384.86**	**1384.86**	1384.95
	bound	-Inf	-Inf	1384.82	1384.82	1384.82	1384.81	**1384.86**	-Inf
	runtime	0.03	3.61	3.15	4.75	3824.62	82.98	0.11	73.29
mean energy		1587.13	1441.43	1694.72	3300.67	2800.54	4007.73	1400.57	1400.68
mean bound		-Inf	-Inf	1400.33	1400.30	1400.35	1392.01	1400.57	-Inf
mean runtime		0.05	18.20	6.36	9.68	3496.50	101.56	0.13	97.37
best value		0.00	10.00	10.00	10.00	10.00	20.00	100.00	40.00
best bound		0.00	0.00	10.00	0.00	0.00	0.00	100.00	0.00
verified opt		0.00	0.00	10.00	0.00	0.00	0.00	100.00	0.00

TABLE B.10 Segmentation.

inclusion		BCD	FW	PGD
modelH-1-0.8-0.2	value	12435.37	7419.38	7421.24
	bound	-Inf	-Inf	-Inf
	runtime	0.14	44.22	67.47
modelH-10-0.8-0.2	value	15446.57	7427.81	5424.26
	bound	-Inf	-Inf	-Inf
	runtime	0.14	2.76	16.90
modelH-2-0.8-0.2	value	10430.00	5425.92	5425.74
	bound	-Inf	-Inf	-Inf
	runtime	0.14	11.55	57.53
modelH-3-0.8-0.2	value	15397.00	1382.80	1382.23
	bound	-Inf	-Inf	-Inf
	runtime	0.14	20.57	19.35
modelH-4-0.8-0.2	value	15447.30	4427.73	4427.66
	bound	-Inf	-Inf	-Inf
	runtime	0.13	8.25	109.47
modelH-5-0.8-0.2	value	9391.02	6385.98	6385.44
	bound	-Inf	-Inf	-Inf
	runtime	0.13	6.26	32.41
modelH-6-0.8-0.2	value	13420.27	5407.69	3403.83
	bound	-Inf	-Inf	-Inf
	runtime	0.14	36.05	24.21
modelH-7-0.8-0.2	value	11438.71	10411.17	11498.09
	bound	-Inf	-Inf	-Inf
	runtime	0.13	18.45	72.97
modelH-8-0.8-0.2	value	14385.72	6376.91	6375.75
	bound	-Inf	-Inf	-Inf
	runtime	0.14	35.24	80.66
modelH-9-0.8-0.2	value	7393.92	3386.31	3385.93
	bound	-Inf	-Inf	-Inf
	runtime	0.14	28.90	29.45
mean energy		12518.59	5805.17	5513.02
mean bound		-Inf	-Inf	-Inf
mean runtime		0.14	21.23	51.04
best value		0.00	0.00	0.00
best bound		0.00	0.00	0.00
verified opt		0.00	0.00	0.00

TABLE B.11 Second-order stereo.

stereo		α-Fusion	TRBP	ADDD	MPLP	MPLP-C	BUNDLE	SRMP	ADMM
art_small	value	13262.49	13336.35	13543.70	NaN	NaN	15105.28	**13091.20**	13297.79
	bound	-Inf	-Inf	12925.76	NaN	NaN	12178.62	**13069.30**	-Inf
	runtime	50.99	3744.91	3096.10	NaN	NaN	3845.89	3603.89	3710.92
cones_small	value	18582.85	18640.25	18763.13	NaN	NaN	20055.65	**18433.01**	18590.87
	bound	-Inf	-Inf	18334.00	NaN	NaN	17724.56	**18414.29**	-Inf
	runtime	48.89	3660.77	7506.15	NaN	NaN	3814.74	3603.11	3659.15
teddy_small	value	14653.53	14680.21	14804.46	NaN	NaN	15733.15	**14528.74**	14715.83
	bound	-Inf	-Inf	14374.12	NaN	NaN	13981.71	**14518.03**	-Inf
	runtime	50.99	3670.35	3535.79	NaN	NaN	3820.05	3603.49	3620.84
venus_small	value	9644.78	9692.80	9796.44	NaN	NaN	9990.68	**9606.34**	9669.62
	bound	-Inf	-Inf	9377.05	NaN	NaN	9402.97	**9601.86**	-Inf
	runtime	49.24	3627.58	3761.29	NaN	NaN	3774.66	3603.14	3657.60
mean energy		14035.91	14087.40	14226.93	NaN	NaN	15221.19	13914.82	14068.53
mean bound		-Inf	-Inf	13752.73	NaN	NaN	13321.96	13900.87	-Inf
mean runtime		50.03	3675.90	4474.83	NaN	NaN	3813.84	3603.41	3662.13
best value		0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00
best bound		0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00
verified opt		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

TABLE B.12 Second-order stereo.

stereo		BCD	FW	PGD
art_small	value	13896.67	13696.50	13929.06
	bound	-Inf	-Inf	-Inf
	runtime	60.63	1407.07	3648.00
cones_small	value	18926.70	18776.26	19060.17
	bound	-Inf	-Inf	-Inf
	runtime	57.40	2111.63	3669.24
teddy_small	value	14998.31	14891.12	15193.23
	bound	-Inf	-Inf	-Inf
	runtime	60.08	1626.66	3671.59

stereo		BCD	FW	PGD
venus_small	value	9767.21	9726.27	9992.13
	bound	-Inf	-Inf	-Inf
	runtime	60.27	1851.40	3670.82
mean energy		14397.22	14272.54	14543.65
mean bound		-Inf	-Inf	-Inf
mean runtime		59.59	1749.19	3664.92
best value		0.00	0.00	0.00
best bound		0.00	0.00	0.00
verified opt		0.00	0.00	0.00

C

Theoretical Proofs and Additional Details for Chapter 7

C.1 PROOFS OF THEORETICAL RESULTS

C.1.1 *Proof of non-differentiability of standard ADMM updates*

In Section 7.5, we claimed that the ADMM updates (7.67) and (7.68) are generally non-differentiable. We have seen in Chapters 5 and 6 that, for MAP inference or one-to-one graph matching, these updates are reduced to projections onto probability simplices. To prove our claim, it suffices to consider the two-dimensional case. We will show that the following function is non-differentiable (with respect to $\boldsymbol{\beta}$):

$$\mathbf{w}^*(\boldsymbol{\beta}) := \operatorname*{argmin}_{\substack{\mathbf{w}\in\mathbb{R}^2_+ \\ \mathbf{1}^\top\mathbf{w}=1}} \|\mathbf{w}-\boldsymbol{\beta}\|_2^2. \tag{C.1}$$

Denote $\mathbf{w} = (w_1, w_2)$ and $\boldsymbol{\beta} = (\beta_1, \beta_2)$. Notice that $w_2 = 1 - w_1$, we can reduce the above minimization problem to a single-variable one with respect to w_1:

$$\min_{0\le w_1\le 1} \left\{(w_1-\beta_1)^2 + (1-w_1-\beta_2)^2\right\}. \tag{C.2}$$

The above is a convex quadratic function. It is straightforward to find its global minimum over $[0, 1]$:

$$w_1^* = \begin{cases} 0 & \text{if } \beta_1-\beta_2 < -1, \\ \frac{1}{2}(\beta_1-\beta_2+1) & \text{if } -1 \le \beta_1-\beta_2 \le 1, \\ 1 & \text{if } 1 < \beta_1-\beta_2. \end{cases} \tag{C.3}$$

Therefore, the optimal solution to (C.1) is given by $\mathbf{w}^* = (w_1^*, w_2^*)$, with $w_2^* = 1 - w_1^*$. Clearly, w_1^* is non-differentiable. More specifically, it is not differentiable at points $\boldsymbol{\beta}$ that satisfy $|\beta_1 - \beta_2| = 1$. We conclude that $\mathbf{w}^*$ is non-differentiable.

Remark. In Figure 7.2a on page 86, we plotted the following function:

$$\mathbf{w}^*(\boldsymbol{\beta}) = \operatorname*{argmin}_{\substack{\mathbf{w}\in\mathbb{R}^2_+ \\ \mathbf{1}^\top\mathbf{w}=1}} \left\{\frac{1}{2}\|\mathbf{w}-\mathbf{1}\|_2^2 - \boldsymbol{\beta}^\top\mathbf{w}\right\}. \tag{C.4}$$

One can easily reduce (C.4) to (C.1) and then use (C.3) to verify that their solutions

are identical.

C.1.2 *Proof of Equations (7.94) and (7.95)*

Rewrite (7.90) as

$$X_{is} = \frac{Z_{is}E_{is}}{S_i} \quad \forall i \in \mathcal{V}, \forall s \in \mathcal{S}. \tag{C.5}$$

It is straightforward to see that:

$$\frac{\partial X_{is}}{\partial Z_{is}} = \frac{E_{is}(S_i - Z_{is}E_{is})}{S_i^2} \qquad \forall i \in \mathcal{V}, \forall s \in \mathcal{S}, \tag{C.6}$$

$$\frac{\partial X_{is}}{\partial Z_{it}} = \frac{-Z_{is}E_{is}E_{it}}{S_i^2} \qquad \forall i \in \mathcal{V}, \forall s, t \in \mathcal{S}, s \neq t, \tag{C.7}$$

$$\frac{\partial X_{is}}{\partial Z_{jt}} = 0 \qquad \forall i, j \in \mathcal{V}, i \neq j, \forall s, t \in \mathcal{S}, \tag{C.8}$$

$$\frac{\partial X_{is}}{\partial V_{is}} = \frac{E_{is}Z_{is}(S_i - E_{is}Z_{is})}{S_i^2} \qquad \forall i \in \mathcal{V}, \forall s \in \mathcal{S}, \tag{C.9}$$

$$\frac{\partial X_{is}}{\partial V_{it}} = \frac{-E_{is}E_{it}Z_{is}Z_{it}}{S_i^2} \qquad \forall i \in \mathcal{V}, \forall s, t \in \mathcal{S}, s \neq t, \tag{C.10}$$

$$\frac{\partial X_{is}}{\partial V_{jt}} = 0 \qquad \forall i, j \in \mathcal{V}, i \neq j, \forall s, t \in \mathcal{S}. \tag{C.11}$$

Therefore, it is clearly that the Jacobian matrices $\frac{\partial \mathbf{x}}{\partial \mathbf{z}}$ and $\frac{\partial \mathbf{x}}{\partial \mathbf{v}}$ are block diagonal matrices whose diagonal blocks are given by (7.94) and (7.95), respectively.

C.2 DETAILED EXPERIMENTAL RESULTS

We provide in Table C.1 the detailed results for the experiments presented in Section 7.7.2.

TABLE C.1 Per-class accuracy of the models on the Pascal VOC 2012 test set.

method	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
FCN	78.1	34.1	76.5	54.6	64.4	84.3	76.9	79.2	30.1	72.7	53.3	73.8	74.2	76.4	79.1	47.7	76.1	48.7	73.5	61.6
MFCRF	81.6	35.5	79.9	56.7	66.2	86.0	79.1	81.6	30.6	75.3	54.0	76.7	76.3	77.8	80.8	49.8	79.3	50.0	75.5	63.6
ADCRF10	81.7	35.6	81.1	58.4	67.8	86.8	79.7	82.4	31.1	77.1	56.5	79.4	80.1	76.8	80.9	52.1	82.4	48.4	77.2	65.2
ADCRF50	82.0	35.7	81.4	58.5	68.0	86.9	79.9	82.6	31.2	77.3	56.6	79.6	80.2	76.9	81.1	52.3	82.7	48.5	77.4	65.3
A-FCN	79.8	36.4	77.6	55.0	63.9	84.6	76.9	80.7	28.3	72.5	53.6	75.6	74.1	79.3	78.6	55.2	75.4	50.1	72.1	60.0
MFCRF	87.0	37.3	77.8	60.6	66.3	88.2	82.8	84.1	31.1	77.0	56.4	79.5	79.2	83.5	80.7	56.4	78.2	43.8	76.5	62.7
ADCRF10	85.6	37.6	80.0	60.0	66.9	88.3	80.8	84.4	30.1	77.3	54.9	79.5	78.2	83.4	79.6	57.2	78.7	54.5	76.3	62.9
ADCRF50	85.8	37.7	80.2	60.0	67.0	88.4	80.9	84.5	30.3	77.4	55.0	79.7	78.4	83.5	79.7	57.3	78.9	54.8	76.4	63.0

Bibliography

[Andres et al., 2012] Andres, B., Beier, T., and Kappes, J. (2012). OpenGM: A C++ library for discrete graphical models. *CoRR*, abs/1206.0111. 66

[Baydin et al., 2018] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2018). Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43. 76, 78, 82, 87

[Belanger et al., 2017] Belanger, D., Yang, B., and McCallum, A. (2017). End-to-end learning for structured prediction energy networks. *arXiv preprint arXiv:1703.05667*. 72

[Belongie et al., 2002] Belongie, S., Malik, J., and Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE transactions on pattern analysis and machine intelligence*, 24(4):509–522. 45

[Bertsekas, 1999] Bertsekas, D. P. (1999). *Nonlinear programming*. Athena scientific Belmont. 3, 33, 35, 54, 58, 59, 64, 108

[Besag, 1986] Besag, J. (1986). On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 259–302. 57

[Bottou et al., 2018] Bottou, L., Curtis, F. E., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311. 74

[Boyd et al., 2011] Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122. 4, 23, 24, 25, 29

[Boyd and Vandenberghe, 2004] Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press. 21

[Boykov et al., 1999] Boykov, Y., Veksler, O., and Zabih, R. (1999). Fast approximate energy minimization via graph cuts. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 377–384. IEEE. 2

[Boykov et al., 2001] Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239. 2, 11, 66

[Brakel et al., 2013] Brakel, P., Stroobandt, D., and Schrauwen, B. (2013). Training energy-based models for time-series imputation. *The Journal of Machine Learning Research*, 14(1):2771–2797. 72

[Bregman, 1967] Bregman, L. M. (1967). The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR computational mathematics and mathematical physics*, 7(3):200–217. 31, 84

[Burkard et al., 1998] Burkard, R. E., Cela, E., Pardalos, P. M., and Pitsoulis, L. S. (1998). The quadratic assignment problem. In *Handbook of combinatorial optimization*, pages 1713–1809. Springer. 3, 15

[Censor and Zenios, 1997] Censor, Y. and Zenios, S. A. (1997). *Parallel optimization: Theory, algorithms, and applications*. Oxford University Press on Demand. 31, 84

[Chen et al., 2016] Chen, C., He, B., Ye, Y., and Yuan, X. (2016). The direct extension of admm for multi-block convex minimization problems is not necessarily convergent. *Mathematical Programming*, 155(1-2):57–79. 26, 27

[Chen et al., 2015] Chen, C., Li, M., Liu, X., and Ye, Y. (2015). Extended admm and bcd for nonseparable convex minimization models with quadratic coupling terms: convergence analysis and insights. *Mathematical Programming*, pages 1–41. 27

[Chen et al., 2017] Chen, L., Sun, D., and Toh, K.-C. (2017). A note on the convergence of admm for linearly constrained convex optimization problems. *Computational Optimization and Applications*, 66(2):327–343. 25

[Chen et al., 2014] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2014). Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*. 92, 93

[Cho et al., 2010] Cho, M., Lee, J., and Lee, K. M. (2010). Reweighted random walks for graph matching. In *Computer Vision–ECCV 2010*, pages 492–505. Springer. 3, 18, 19, 44, 46

[Cho et al., 2014] Cho, M., Sun, J., Duchenne, O., and Ponce, J. (2014). Finding matches in a haystack: A max-pooling strategy for graph matching in the presence of outliers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2083–2090. 19, 44

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. `https://keras.io`. 94

[Condat, 2016] Condat, L. (2016). Fast projection onto the simplex and the ℓ_1 ball. *Mathematical Programming*, 158(1-2):575–585. 42, 59, 64, 103

[Cour et al., 2007] Cour, T., Srinivasan, P., and Shi, J. (2007). Balanced graph matching. *Advances in Neural Information Processing Systems*, 19:313. 3, 18, 44

[Cui et al., 2015] Cui, Y., Li, X., Sun, D., and Toh, K.-C. (2015). On the convergence properties of a majorized admm for linearly constrained convex optimization problems with coupled objective functions. *arXiv preprint arXiv:1502.00098*. 27

[Davis and Yin, 2016] Davis, D. and Yin, W. (2016). Convergence rate analysis of several splitting schemes. In *Splitting Methods in Communication, Imaging, Science, and Engineering*, pages 115–163. Springer. 25

[Deng and Yin, 2016] Deng, W. and Yin, W. (2016). On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing*, 66(3):889–916. 25, 31

[Domke, 2011] Domke, J. (2011). Parameter learning with truncated message-passing. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2937–2943. IEEE. 72

[Domke, 2012] Domke, J. (2012). Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pages 318–326. 72, 76, 92

[Duchenne et al., 2011] Duchenne, O., Bach, F., Kweon, I.-S., and Ponce, J. (2011). A tensor-based algorithm for high-order graph matching. *IEEE transactions on pattern analysis and machine intelligence*, 33(12):2383–2395. 3, 15, 19, 44, 48

[Eckstein, 1994] Eckstein, J. (1994). Parallel alternating direction multiplier decomposition of convex programs. *Journal of Optimization Theory and Applications*, 80(1):39–62. 31

[Eckstein and Bertsekas, 1992] Eckstein, J. and Bertsekas, D. P. (1992). On the douglas—rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1-3):293–318. 31

[Everingham et al., 2010] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338. 91, 93

[Faugeras, 1993] Faugeras, O. (1993). *Three-dimensional computer vision: a geometric viewpoint*. MIT Press. 76

[Fix et al., 2011] Fix, A., Gruber, A., Boros, E., and Zabih, R. (2011). A graph cut algorithm for higher-order markov random fields. In *2011 International Conference on Computer Vision*, pages 1020–1027. IEEE. 2, 66

[Fix et al., 2014] Fix, A., Wang, C., and Zabih, R. (2014). A primal-dual algorithm for higher-order multilabel markov random fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1138–1145. 12

[Frank and Wolfe, 1956] Frank, M. and Wolfe, P. (1956). An algorithm for quadratic programming. *Naval Research Logistics (NRL)*, 3(1-2):95–110. 54

[Frey and MacKay, 1997] Frey, B. J. and MacKay, D. J. C. (1997). A revolution: Belief propagation in graphs with cycles. In *In Neural Information Processing Systems*, pages 479–485. MIT Press. 11

[Gabay and Mercier, 1975] Gabay, D. and Mercier, B. (1975). *A dual algorithm for the solution of non linear variational problems via finite element approximation*. Institut de recherche d'informatique et d'automatique. 3, 23

[Gao and Zhang, 2017] Gao, X. and Zhang, S.-Z. (2017). First-order algorithms for convex optimization with nonseparable objective and coupled constraints. *Journal of the Operations Research Society of China*, 5(2):131–159. 27

[Geman and Geman, 1987] Geman, S. and Geman, D. (1987). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. In *Readings in Computer Vision*, pages 564–584. Elsevier. 2

[Giselsson, 2017] Giselsson, P. (2017). Tight global linear convergence rate bounds for douglas–rachford splitting. *Journal of Fixed Point Theory and Applications*, 19(4):2241–2270. 25

[Globerson and Jaakkola, 2008] Globerson, A. and Jaakkola, T. S. (2008). Fixing max-product: Convergent message passing algorithms for map lp-relaxations. In *Advances in neural information processing systems*, pages 553–560. 2, 66

[Glowinski and Marroco, 1975] Glowinski, R. and Marroco, A. (1975). Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires. *Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique*, 9(R2):41–76. 3, 23

[Gold and Rangarajan, 1996] Gold, S. and Rangarajan, A. (1996). A graduated assignment algorithm for graph matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(4):377–388. 3, 18

[Gonçalves et al., 2017] Gonçalves, M. L., Melo, J. G., and Monteiro, R. D. (2017). Convergence rate bounds for a proximal admm with over-relaxation stepsize parameter for solving nonconvex linearly constrained problems. *arXiv preprint arXiv:1702.01850*. 28, 29, 31

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. 74

[Gould et al., 2016] Gould, S., Fernando, B., Cherian, A., Anderson, P., Cruz, R. S., and Guo, E. (2016). On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv preprint arXiv:1607.05447*. 76, 78

[Griewank, 2010] Griewank, A. (2010). Who invented the reverse mode of differentiation? *Documenta Mathematica*. 80, 82

[Guo et al., 2017] Guo, K., Han, D., and Wu, T.-T. (2017). Convergence of alternating direction method for minimizing sum of two nonconvex functions with linear constraints. *International Journal of Computer Mathematics*, 94(8):1653–1669. 28, 29, 31

[Han and Yuan, 2012] Han, D. and Yuan, X. (2012). A note on the alternating direction method of multipliers. *Journal of Optimization Theory and Applications*, 155(1):227–238. 26

[Hariharan et al., 2014] Hariharan, B., Arbeláez, P., Girshick, R., and Malik, J. (2014). Simultaneous detection and segmentation. In *European Conference on Computer Vision*, pages 297–312. Springer. 93

[He et al., 2002] He, B., Liao, L.-Z., Han, D., and Yang, H. (2002). A new inexact alternating directions method for monotone variational inequalities. *Mathematical Programming*, 92(1):103–118. 31

[He et al., 2000] He, B., Yang, H., and Wang, S. (2000). Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities. *Journal of Optimization Theory and applications*, 106(2):337–356. 29

[He and Yuan, 2015] He, B. and Yuan, X. (2015). On non-ergodic convergence rate of douglas–rachford alternating direction method of multipliers. *Numerische Mathematik*, 130(3):567–577. 25

[Hendrik Kappes et al., 2013] Hendrik Kappes, J., Speth, M., Reinelt, G., and Schnorr, C. (2013). Towards efficient and exact map-inference for large scale discrete computer vision problems via combinatorial optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1752–1758. 12

[Hong et al., 2014] Hong, M., Chang, T.-H., Wang, X., Razaviyayn, M., Ma, S., and Luo, Z.-Q. (2014). A block successive upper bound minimization method of multipliers for linearly constrained convex optimization. *arXiv preprint arXiv:1401.7079*. 27

[Hong and Luo, 2017] Hong, M. and Luo, Z.-Q. (2017). On the linear convergence of the alternating direction method of multipliers. *Mathematical Programming*, 162(1-2):165–199. 25

[Hong et al., 2016] Hong, M., Luo, Z.-Q., and Razaviyayn, M. (2016). Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. *SIAM Journal on Optimization*, 26(1):337–364. 27, 28, 29

[Jiang et al., 2016] Jiang, B., Lin, T., Ma, S., and Zhang, S. (2016). Structured nonconvex and nonsmooth optimization: algorithms and iteration complexity analysis. *arXiv preprint arXiv:1605.02408*. 28

[Jojic et al., 2010] Jojic, V., Gould, S., and Koller, D. (2010). Accelerated dual decomposition for map inference. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 503–510. 12

[Kappes et al., 2015] Kappes, J. H., Andres, B., Hamprecht, F. A., Schnörr, C., Nowozin, S., Batra, D., Kim, S., Kausler, B. X., Kröger, T., Lellmann, J., Komodakis, N., Savchynskyy, B., and Rother, C. (2015). A comparative study of modern inference techniques for structured discrete energy minimization problems. *International Journal of Computer Vision*, pages 1–30. 11, 67

[Kappes et al., 2012] Kappes, J. H., Savchynskyy, B., and Schnörr, C. (2012). A bundle approach to efficient map-inference by lagrangian relaxation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1688–1695. IEEE. 12, 66

[Kappes et al., 2011] Kappes, J. H., Speth, M., Andres, B., Reinelt, G., and Schn, C. (2011). Globally optimal image partitioning by multicuts. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 31–44. Springer. 12

[Kappes et al., 2016] Kappes, J. H., Speth, M., Reinelt, G., and Schnörr, C. (2016). Higher-order segmentation via multicuts. *Computer Vision and Image Understanding*, 143:104–119. 12

[Koller and Friedman, 2009] Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press. 5, 7, 71

[Kolmogorov, 2006] Kolmogorov, V. (2006). Convergent tree-reweighted message passing for energy minimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1568–1583. 2, 11, 12, 66

[Kolmogorov, 2015] Kolmogorov, V. (2015). A new look at reweighted message passing. *IEEE transactions on pattern analysis and machine intelligence*, 37(5):919–930. 66

[Komodakis et al., 2011] Komodakis, N., Paragios, N., and Tziritas, G. (2011). Mrf energy minimization and beyond via dual decomposition. *IEEE transactions on pattern analysis and machine intelligence*, 33(3):531–552. 2, 3, 12, 33, 66

[Komodakis et al., 2008] Komodakis, N., Tziritas, G., and Paragios, N. (2008). Performance vs computational efficiency for optimizing single and dynamic mrfs: Setting the state of the art with primal-dual strategies. *Computer Vision and Image Understanding*, 112(1):14–29. 2, 12, 66

[Koopmans and Beckmann, 1957] Koopmans, T. C. and Beckmann, M. (1957). Assignment problems and the location of economic activities. *Econometrica: journal of the Econometric Society*, pages 53–76. 19

[Krähenbühl and Koltun, 2011] Krähenbühl, P. and Koltun, V. (2011). Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in neural information processing systems*, pages 109–117. 92

[Krähenbühl and Koltun, 2013] Krähenbühl, P. and Koltun, V. (2013). Parameter learning and convergent inference for dense random fields. In *International Conference on Machine Learning*, pages 513–521. 72, 76, 92, 93, 96

[Kschischang et al., 2001] Kschischang, F. R., Frey, B. J., and Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519. 9

[Kuhn, 1955] Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97. 14, 18, 35

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86. 85

[Kumar et al., 2009] Kumar, M. P., Kolmogorov, V., and Torr, P. H. S. (2009). An analysis of convex relaxations for map estimation of discrete mrfs. *J. Mach. Learn. Res.*, 10:71–106. 12

[Lawler, 1963] Lawler, E. L. (1963). The quadratic assignment problem. *Management science*, 9(4):586–599. 15

[Lê-Huu and Paragios, 2017] Lê-Huu, D. K. and Paragios, N. (2017). Alternating direction graph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4914–4922. 33

[Lê-Huu and Paragios, 2018] Lê-Huu, D. K. and Paragios, N. (2018). Continuous relaxation of map inference: A nonconvex perspective. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 53

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551. 82

[Lee et al., 2011] Lee, J., Cho, M., and Lee, K. M. (2011). Hyper-graph matching via reweighted random walks. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1633–1640. IEEE. 3, 19, 44

[Leordeanu and Hebert, 2005] Leordeanu, M. and Hebert, M. (2005). A spectral technique for correspondence problems using pairwise constraints. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1482–1489. IEEE. 3, 18, 19, 44

[Leordeanu et al., 2009] Leordeanu, M., Hebert, M., and Sukthankar, R. (2009). An integer projected fixed point method for graph matching and map inference. In *Advances in neural information processing systems*, pages 1114–1122. 3, 18, 44

[Leordeanu et al., 2012] Leordeanu, M., Sukthankar, R., and Hebert, M. (2012). Unsupervised learning for graph matching. *International journal of computer vision*, 96(1):28–45. 48, 49

[Li and Pong, 2015] Li, G. and Pong, T. K. (2015). Global convergence of splitting methods for nonconvex composite optimization. *SIAM Journal on Optimization*, 25(4):2434–2460. 28, 29, 31

[Lin et al., 2015] Lin, T.-Y., Ma, S.-Q., and Zhang, S.-Z. (2015). On the sublinear convergence rate of multi-block admm. *Journal of the Operations Research Society of China*, 3(3):251–274. 27

[Linnainmaa, 1970] Linnainmaa, S. (1970). The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, pages 6–7. 82

[Long et al., 2015] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440. 92, 93

[Martins et al., 2015] Martins, A. F., Figueiredo, M. A., Aguiar, P. M., Smith, N. A., and Xing, E. P. (2015). Ad3: Alternating directions dual decomposition for map inference in graphical models. *Journal of Machine Learning Research*, 16:495–545. 3, 12, 33, 66

[Monteiro et al., 2018] Monteiro, M., Figueiredo, M. A., and Oliveira, A. L. (2018). Conditional random fields as recurrent neural networks for 3d medical imaging segmentation. *arXiv preprint arXiv:1807.07464*. 94

[Nesterov, 2013] Nesterov, Y. (2013). *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media. 22

[Nguyen et al., 2015] Nguyen, Q., Gautier, A., and Hein, M. (2015). A flexible tensor block coordinate ascent scheme for hypergraph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5270–5278. 3, 19, 34, 44, 46

[Nishihara et al., 2015] Nishihara, R., Lessard, L., Recht, B., Packard, A., and Jordan, M. I. (2015). A general analysis of the convergence of admm. *arXiv preprint arXiv:1502.02009*. 31

[Nowozin et al., 2011] Nowozin, S., Lampert, C. H., et al. (2011). Structured learning and prediction in computer vision. *Foundations and Trends® in Computer Graphics and Vision*, 6(3–4):185–365. 71

[Olsson et al., 2007] Olsson, C., Eriksson, A. P., and Kahl, F. (2007). Solving large scale binary quadratic problems: Spectral methods vs. semidefinite programming. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE. 2

[Otten and Dechter, 2012] Otten, L. and Dechter, R. (2012). Anytime and/or depth-first search for combinatorial optimization. *AI Communications*, 25(3):211–227. 12

[Pearl, 1982] Pearl, J. (1982). Reverend bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the Second AAAI Conference on Artificial Intelligence*, AAAI'82, pages 133–136. AAAI Press. 11

[Ravikumar and Lafferty, 2006] Ravikumar, P. and Lafferty, J. (2006). Quadratic programming relaxations for metric labeling and markov random field map estimation. In *Proceedings of the 23rd international conference on Machine learning*, pages 737–744. ACM. 2, 12, 53, 58, 66, 69, 112

[Ross et al., 2011] Ross, S., Munoz, D., Hebert, M., and Bagnell, J. A. (2011). Learning message-passing inference machines for structured prediction. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2737–2744. IEEE. 72

[Rother et al., 2007] Rother, C., Kolmogorov, V., Lempitsky, V., and Szummer, M. (2007). Optimizing binary mrfs via extended roof duality. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE. 11

[Sahni and Gonzalez, 1976] Sahni, S. and Gonzalez, T. (1976). P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565. 15

[Savchynskyy et al., 2013] Savchynskyy, B., Kappes, J. H., Swoboda, P., and Schnörr, C. (2013). Global map-optimality by shrinking the combinatorial search area with convex relaxation. In *Advances in Neural Information Processing Systems*, pages 1950–1958. 12

[Savchynskyy et al., 2011] Savchynskyy, B., Schmidt, S., Kappes, J., and Schnorr, C. (2011). A study of nesterov's scheme for lagrangian decomposition and map labeling.

In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1817–1823. IEEE. 12

[Scharstein and Szeliski, 2003] Scharstein, D. and Szeliski, R. (2003). High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE. 67, 69

[Schrijver, 2002] Schrijver, A. (2002). *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media. 43

[Shimony, 1994] Shimony, S. E. (1994). Finding maps for belief networks is np-hard. *Artificial Intelligence*, 68(2):399–410. 2, 11

[Shor et al., 1985] Shor, N., Kiwiel, K., and Ruszcaynski, A. (1985). *Minimization methods for non-differentiable functions*. Springer-Verlag New York, Inc. 22, 35

[Shotton et al., 2009] Shotton, J., Winn, J., Rother, C., and Criminisi, A. (2009). Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *International Journal of Computer Vision*, 81(1):2–23. 92

[Sontag et al., 2012] Sontag, D., Li, Y., et al. (2012). Efficiently searching for frustrated cycles in map inference. In *28th Conference on Uncertainty in Artificial Intelligence, UAI 2012*. 66

[Stoyanov et al., 2011] Stoyanov, V., Ropson, A., and Eisner, J. (2011). Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 725–733. 72

[Tappen et al., 2007] Tappen, M. F., Liu, C., Adelson, E. H., and Freeman, W. T. (2007). Learning gaussian conditional random fields for low-level vision. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE. 76

[Tatikonda and Jordan, 2002] Tatikonda, S. C. and Jordan, M. I. (2002). Loopy belief propagation and gibbs measures. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 493–500. Morgan Kaufmann Publishers Inc. 72

[Themelis and Patrinos, 2017] Themelis, A. and Patrinos, P. (2017). Douglas-Rachford splitting and ADMM for nonconvex optimization: tight convergence results. *ArXiv e-prints*. 28, 29, 31

[Torresani et al., 2013] Torresani, L., Kolmogorov, V., and Rother, C. (2013). A dual decomposition approach to feature correspondence. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(2):259–271. 3, 18, 19, 33, 35, 44, 45, 46, 48, 49

[Vapnik, 1992] Vapnik, V. (1992). Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838. 73

[Wainwright et al., 2005] Wainwright, M. J., Jaakkola, T. S., and Willsky, A. S. (2005). Map estimation via agreement on trees: message-passing and linear programming. *IEEE transactions on information theory*, 51(11):3697–3717. 2, 11, 12, 55, 66

[Waki et al., 2006] Waki, H., Kim, S., Kojima, M., and Muramatsu, M. (2006). Sums of squares and semidefinite program relaxations for polynomial optimization problems with structured sparsity. *SIAM Journal on Optimization*, 17(1):218–242. 12

[Wang et al., 2013] Wang, C., Komodakis, N., and Paragios, N. (2013). Markov random field modeling, inference & learning in computer vision & image understanding: A survey. *Computer Vision and Image Understanding*, 117(11):1610–1627. 11

[Wang et al., 2015a] Wang, F., Cao, W., and Xu, Z. (2015a). Convergence of multi-block bregman admm for nonconvex composite problems. *arXiv preprint arXiv:1505.03063*. 28, 31

[Wang and Banerjee, 2013] Wang, H. and Banerjee, A. (2013). Online alternating direction method (longer version). *arXiv preprint arXiv:1306.3721*. 25

[Wang and Banerjee, 2014] Wang, H. and Banerjee, A. (2014). Bregman alternating direction method of multipliers. In *Advances in Neural Information Processing Systems*, pages 2816–2824. 31, 84

[Wang and Liao, 2001] Wang, S. and Liao, L. (2001). Decomposition method with a variable parameter for a class of monotone variational inequality problems. *Journal of optimization theory and applications*, 109(2):415–429. 30

[Wang et al., 2015b] Wang, Y., Yin, W., and Zeng, J. (2015b). Global convergence of admm in nonconvex nonsmooth optimization. *arXiv preprint arXiv:1511.06324*. 29, 30, 65

[Woodford et al., 2009] Woodford, O., Torr, P., Reid, I., and Fitzgibbon, A. (2009). Global stereo reconstruction under second-order smoothness priors. *IEEE transactions on pattern analysis and machine intelligence*, 31(12):2115–2128. 67

[Xu et al., 2017] Xu, Z., Figueiredo, M. A., Yuan, X., Studer, C., and Goldstein, T. (2017). Adaptive relaxed admm: Convergence theory and practical implementation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7389–7398. 30

[Yedidia et al., 2005] Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2005). Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312. 2

[Zaslavskiy et al., 2009] Zaslavskiy, M., Bach, F., and Vert, J.-P. (2009). A path following algorithm for the graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2227–2242. 18

[Zass and Shashua, 2008] Zass, R. and Shashua, A. (2008). Probabilistic graph and hypergraph matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE. 3, 19, 44

[Zeng et al., 2010] Zeng, Y., Wang, C., Wang, Y., Gu, X., Samaras, D., and Paragios, N. (2010). Dense non-rigid surface registration using high-order graph matching. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 382–389. IEEE. 19, 33

[Zheng et al., 2015] Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., and Torr, P. H. (2015). Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1529–1537. 72, 92, 93, 94, 95, 96

[Zhou and De la Torre, 2012] Zhou, F. and De la Torre, F. (2012). Factorized graph matching. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 127–134. IEEE. 18, 49

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Title **Nonconvex Alternating Direction Optimization for Graphs: Inference and Learning.**

Keywords ADMM, graph matching, Markov random fields, graphical models, inference, learning.

Abstract This thesis presents our contributions to inference and learning of graph-based models in computer vision. First, we propose a novel class of decomposition algorithms for solving graph and hypergraph matching based on the nonconvex alternating direction method of multipliers (ADMM). These algorithms are computationally efficient and highly parallelizable. Furthermore, they are also very general and can be applied to arbitrary energy functions as well as arbitrary assignment constraints. Experiments show that they outperform existing state-of-the-art methods on popular benchmarks. Second, we propose a nonconvex continuous relaxation of maximum a posteriori (MAP) inference in discrete Markov random fields (MRFs). We show that this relaxation is tight for arbitrary MRFs. This allows us to apply continuous optimization techniques to solve the original discrete problem without loss in accuracy after rounding. We study two popular gradient-based methods, and further propose a more effective solution using nonconvex ADMM. Experiments on different real-world problems demonstrate that the proposed ADMM compares favorably with state-of-the-art algorithms in different settings. Finally, we propose a method for learning the parameters of these graph-based models from training data, based on nonconvex ADMM. This method consists of viewing ADMM iterations as a sequence of differentiable operations, which allows efficient computation of the gradient of the training loss with respect to the model parameters, enabling efficient training using stochastic gradient descent. At the end we obtain a unified framework for inference and learning with nonconvex ADMM. Thanks to its flexibility, this framework also allows training jointly end-to-end a graph-based model with another model such as a neural network, thus combining the strengths of both. We present experiments on a popular semantic segmentation dataset, demonstrating the effectiveness of our method.

Titre **L'algorithme des directions alternées non convexe pour graphes : inférence et apprentissage.**

Mots clés directions alternées, appariement de graphs, champ aléatoire de Markov, modèles graphiques, inférence, apprentissage.

Résumé Cette thèse présente nos contributions à l'inférence et l'apprentissage des modèles graphiques en vision artificielle. Tout d'abord, nous proposons une nouvelle classe d'algorithmes de décomposition pour résoudre le problème d'appariement de graphes et d'hypergraphes, s'appuyant sur l'algorithme des directions alternées (ADMM) non convexe. Ces algorithmes sont efficaces en terme de calcul et sont hautement parallélisables. En outre, ils sont également très généraux et peuvent être appliqués à des fonctionnelles d'énergie arbitraires ainsi qu'à des contraintes de correspondance arbitraires. Les expériences montrent qu'ils surpassent les méthodes de pointe existantes sur des benchmarks populaires. Ensuite, nous proposons une relaxation continue non convexe pour le problème d'estimation du maximum a posteriori (MAP) dans les champs aléatoires de Markov (MRFs). Nous démontrons que cette relaxation est serrée, c'est-à-dire qu'elle est équivalente au problème original. Cela nous permet d'appliquer des méthodes d'optimisation continue pour résoudre le problème initial discret sans perte de précision après arrondissement. Nous étudions deux méthodes de gradient populaires, et proposons en outre une solution plus efficace utilisant l'ADMM non convexe. Les expériences sur plusieurs problèmes réels démontrent que notre algorithme prend l'avantage sur ceux de pointe, dans différentes configurations. Finalement, nous proposons une méthode d'apprentissage des paramètres de ces modèles graphiques avec des données d'entraînement, basée sur l'ADMM non convexe. Cette méthode consiste à visualiser les itérations de l'ADMM comme une séquence d'opérations différenciables, ce qui permet de calculer efficacement le gradient de la perte d'apprentissage par rapport aux paramètres du modèle. L'apprentissage peut alors utiliser une descente de gradient stochastique. Nous obtenons donc un framework unifié pour l'inférence et l'apprentissage avec l'ADMM non-convexe. Grâce à sa flexibilité, ce framework permet également d'entraîner conjointement de-bout-en-bout un modèle graphique avec un autre modèle, tel qu'un réseau de neurones, combinant ainsi les avantages des deux. Nous présentons des expériences sur un jeu de données de segmentation sémantique populaire, démontrant l'efficacité de notre méthode.

Université Paris-Saclay
Espace Technologique – Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 – 91190 Saint-Aubin, France